

;login:

The USENIX Association Newsletter

Volume 11, Number 4

July/August 1986

CONTENTS

New Executive Director Appointed	3
Contributions to ;login:	3
Call for Papers – Third USENIX Computer Graphics Workshop	3
EUUG Autumn 1986 Workshop	4
Computer Graphics Workshop Proceedings Available	4
Future Meetings	5
Atlanta Tapes to be Available Soon	5
Publications Available	6
T-Shirts	6
Third Annual USENIX Computer Go Tournament	7
The VAX and I	8
<i>Jin Mazumdar</i>	
4.3BSD Manuals	9
HoneyDanBer UUCP – Bringing UNIX® Systems into the Information Age	10
<i>Bill Rieken and Jim Webb</i>	
Netnews Under VM/CMS	36
<i>Irwin Tillman and Peter Honeyman</i>	
Reflections On A UNIX® Scheduler	39
<i>Thomas A. Bohannon</i>	
Local User Groups	46

The closing date for submissions for the next issue of ;login: is August 29, 1986



THE PROFESSIONAL AND TECHNICAL UNIX® ASSOCIATION

;login:

NOTICE

;login: is the official newsletter of the USENIX Association, and is sent free of charge to all members of the Association.

The USENIX Association is an organization of AT&T licensees, sub-licensees, and other persons formed for the purpose of exchanging information and ideas about UNIX[†] and similar operating systems and the C programming language. It is a non-profit corporation incorporated under the laws of the State of Delaware. The officers of the Association are:

President	Alan G. Nemeth
Vice-President	Deborah K. Scherrer
Secretary	Waldo M. Wedel
Treasurer	Stephen C. Johnson
Directors	Rob Kolstad Marshall Kirk McKusick John S. Quarterman David Yost
Executive Director	Peter H. Salus

The editorial staff of *;login:* is:

Editor and Publisher	Peter H. Salus usenix!peter
Copy Editor	Michelle Peetz {masscomp,usenix}!mmp
Production Editor	Tom Strong usenix!tom
Technical Advisor	Rob Kolstad {convex,usenix}!kolstad

Other staff members are:

Betty J. Madden	Office Manager
Emma Reed	Membership Secretary
Jordan Hayes	Technical Consultant

USENIX Association Office
P.O. Box 7
El Cerrito, CA 94530
(415) 528-8649
{ucbvax,decvax}!usenix!office

Judith F. DesHarnais Conference Coordinator

USENIX Conference Office
P.O. Box 385
Sunset Beach, CA 90742
(213) 592-3243 or 592-1381
{ucbvax,decvax}!usenix!judy

John L. Donnelly Exhibit Manager

USENIX Exhibit Office
Oak Bay Building
4750 Table Mesa Drive
Boulder, CO 80303
(303) 499-2600
{ucbvax,decvax}!usenix!john

Contributions Solicited

Members of the UNIX community are heartily encouraged to contribute articles and suggestions for *;login:*. Your contributions may be sent to the editors electronically at the addresses above or through the U.S. mail to the Association office. The USENIX Association reserves the right to edit submitted material.

;login: is produced on UNIX systems using *troff* and a variation of the *-me* macros. We appreciate receiving your contributions in *n/troff* input format, using any macro package. If you contribute hardcopy articles please leave left and right margins of 1" and a top margin of 1½" and a bottom margin of 1¼". Hardcopy output from a line printer or most dot-matrix printers is not reproducible.

Acknowledgments

The Association uses a VAX[‡] 11/730 donated by the Digital Equipment Corporation for support of office and membership functions, preparation of *;login:*, and other association activities. It runs 4.2BSD, which was contributed, installed, and is maintained by mt Xinu. The VAX uses a sixteen line VMZ-32 terminal multiplexor donated by Able Computer of Irvine, California.

Connected to the VAX is a QMS Lasergrafix^{*} 800 Printer System donated by Quality Micro Systems of Mobile, Alabama. It is used for general printing and draft production of *;login:* with *ditroff* software provided by mt Xinu.

This newsletter may contain information covered by one or more licenses, copyrights, and/or non-disclosure agreements. No reproduction of this newsletter in its entirety or in part may be made without written permission of the Association.

[†]UNIX is a registered trademark of AT&T.

[‡]VAX is a trademark of Digital Equipment Corporation.

^{*}Lasergrafix is a trademark of Quality Micro Systems.

New Executive Director Appointed

After a year in office, Jim Ferguson resigned his post as Executive Director of the USENIX Association. At its March meeting in Napa, the Board of Directors appointed Tom Ferrin, Steve Johnson, and Wally Wedel to conduct a search for a new Executive Director. During Tom's absence at the EUUG in Florence, Debbie Scherrer served in his stead.

The USENIX Association Board of Directors is now pleased to announce the appointment of Peter H. Salus as Executive Director. Peter has a BS in Chemistry, an MA in Germanic Languages, and a PhD in Linguistics. Peter has 22 years of academic experience and several years in the business world.

Peter's responsibilities will include supervision of office services, management of the office staff, coordination of conference planning, supervision of the newsletter and workshop proceedings, tape distribution, and other member services. Peter will also handle public relations and membership.

Many members met Peter at Atlanta, where his appointment was announced. He can be reached electronically at {ucbvax,decvax}!usenix!peter.

The Board of Directors

Contributions to ;login:

"Send us your tired, your poor, your huddled ..." manuscripts yearning to be published.

With apologies to Emma Lazarus, as the new Executive Director I would like to encourage all members of the USENIX community to contribute articles, notes, letters or comments to ;login:.

As the Technical and Professional Association of UNIX users, we would like to see ;login: develop into a true professional journal: perhaps eventually splitting into a newsletter plus a journal. In an attempt at raising ;login: to something like a refereed journal, Rob Kolstad will act as technical advisor to me. Sometime during the next few months, USENIX

hopes to make this a regular, part-time post and to appoint a permanent Technical Advisor.

Note that there are three technical articles as well as a piece of humor and Association news and information in this issue. The quantity and quality of ;login:'s contents are a function of the quality of submissions. Please send contributions via e-mail to usenix!login. Please send any graphics to the office by US mail, in original form.

Your contributions should be in n/troff input format, using any macro package.

Peter H. Salus
Executive Director

Call for Papers – Third USENIX Computer Graphics Workshop

The third USENIX Computer Graphics Workshop will be held at the DoubleTree Hotel in Monterey, CA, November 20-21, 1986.

The Workshop will be structured to facilitate in-depth discussions of technical issues with presentations in a variety of formats and ample time for questions and responses. There will be a computer graphics film and video presentation.

Presentations may be from 20 to 40 minutes in length and speakers are encouraged to include visual examples in the media of their choice.

Deadline for submissions is September 15, 1986. Submissions must consist of camera-ready or electronic copy of the paper to be included in the proceedings. Send all materials to:

Reidar J. Bornholdt
Room 7-444
Columbia University
College of Physicians and Surgeons
630 West 168 Street
New York, NY 10032

{harpo|cmc12}!cucard!reidar
{ucbvax|decvax}!usenix!reidar

;login:

EUUG Autumn 1986 Workshop

Distributed UNIX Systems

Manchester, England
22nd-25th September, 1986

The EUUG's Autumn 1986 Workshop is designed to provide a very lively and extremely interesting event which combines a Conference on "Distributed UNIX Systems" with a comprehensive, compact table-top Exhibition - the accent at both being on technical discussions.

The Plenary Sessions will be held daily from 22nd-24th September and Tutorials will be held on 25th September.

Papers have been solicited on Design and Implementation of Distributed UNIX Systems; Mechanisms for Distributing UNIX Functions and Services; Interprocess Communication; Networking; Remote File Systems; Loosely-coupled UNIX Systems; Multiprocessor Systems; Distributed Applications - and many more topics.

Papers are expected from leading UNIX personalities in Israel, Austria, Germany, France, UK, USA, etc., and it is hoped to include presentations from:

Avadis Tevanlan (Carnegie Mellon University) on
"The Mach Project"

Bob Sidebothan (CMU) on

"The Large Scale Distributed File System Project"

Lorie Grob and Jim Litkis (New York University) on

"The Ultra Computer Project"

Tutorial papers have been invited from: Dennis M. Ritchie (Bell Labs); Peter Langston (Bellcore); Kirk McKusick and Mike Karels (UC Berkeley); and in addition UEL (UNIX Europe Ltd) have been invited to talk about streams, RFS, and System V, Release 3.

Costs to include accommodation and ALL meals (exclusive of 15% VAT tax):

Workshop: 200 UK pounds (members)
300 UK pounds (nonmembers)
Tutorials: 100 UK pounds (members)

For further details:

EUUG
Owles Hall
Buntingford, Herts SG9 9PL, UK
+44 44 763 73039
mcvox!ukc!inset!euug

Computer Graphics Workshop Proceedings Available

Copies of the Proceedings of the Second Computer Graphics Workshop at Monterey in December, 1985, are available from the Office for \$3, plus \$7 for overseas airmail postage (prepaid). The following papers are in the proceedings:

"A Dataflow Environment for Interactive Graphics,"
Paul Haeberli

"A Low-Cost Graphics Workstation,"
Spencer Thomas

"MacMix: Mixing Music with a Mouse,"
Adrian Freed

"A Modular Rendering and Modelling System,"
Carlo H. Sequin

"Algorithms for Anti-aliased Rendering,"
Tom Duff

"Scattered Thoughts on Color,"
Roy Hall

"Constructing Uniform Polyhedra,"
Andrew Hume

Copies of the Proceedings of the First Computer Graphics Workshop at Monterey in December, 1984, are also available for \$3, plus \$7 for overseas airmail postage (prepaid).

Future Meetings

USENIX 1987 Winter Conference and UniForum January 20-23, 1987, Washington, DC

The USENIX 1987 Winter Conference will be held at the Shoreham Hotel in Washington, DC, on January 20-23. It will be concurrent with UniForum 1987, which will be at the Washington Convention Center.

The Conference will feature tutorials and three one-day technical sessions:

- Wednesday, January 21: *What is UNIX?*
D. Tilbrook, chair
- Thursday, January 22: *Performance*
H. Schwetman, chair
- Friday, January 23: *Data Bases*
P. Hawthorn, chair

USENIX 1987 Summer Conference and Exhibition Phoenix

The USENIX 1987 Summer Conference and Exhibition will be held on June 8-12, 1987, at the Hyatt Regency Hotel in Phoenix, Arizona. There will be a conference, tutorials, and vendor exhibits.

USENIX 1988 Winter Conference and UniForum Dallas

The USENIX 1988 Winter Conference will be held on February 10-12, 1988, at the Registry Hotel in Dallas, Texas. It will be concurrent with UniForum 1988, which will also be in Dallas. The Conference will feature tutorials and technical sessions.

USENIX 1988 Summer Conference and Exhibition San Francisco

The USENIX 1988 Summer Conference and Exhibition will be held on June 21-24, 1988, at the Hilton Hotel in San Francisco, California. There will be a conference, tutorials, and vendor exhibits.

Long-term USENIX Conference Schedule

Winter '87	Shoreham Hotel, Washington DC	Jan 20-23
Summer '87	Hyatt Regency, Phoenix, AZ	Jun 8-12
Winter '88	Registry Hotel, Dallas, TX	Feb 10-12
Summer '88	Hilton Hotel, San Francisco	Jun 21-24
Summer '89	Hyatt Regency, Baltimore, MD	Jun 13-16
Summer '90	Marriott Hotel, Anaheim, CA	Jun 11-15

The locations and dates for the 1989 and 1990 winter meetings have not yet been fixed.

Atlanta Tapes to be Available Soon

The overwhelming response to the three papers which opened the Atlanta meeting has led USENIX to make videotapes of those presentations available. For those of you who missed these brilliant and original works, they were:

Jon Bentley, Pictures of Programs – a talk on the merits and insights of graphic presentation of algorithmic problems, including the greedy travelling salesman and wire wrapping.

Michael Hawley, MIDI Music Software for UNIX – a vivid presentation of 500 years of musical history with some remarkable examples.

Peter S. Langston, (201) 644-2332 or Eedie & Eddie on the wire: An Experiment in Music Generation – a Dectalk-narrated mathematical approach to computer music generation.

We are currently negotiating with several companies concerning reproduction and merchandising of the videotapes. We are hoping to make them available at cost + postage and handling within a few months. There will be VHS and Beta versions.

If time/space permits, the Awards ceremonies will also be included on the tapes, enabling non-attendees to learn what "e-Chernobyl" means.

Peter H. Salus
Executive Director

Publications Available

The following publications are available from the Association Office or the source indicated. Prices and overseas postage charges are per copy. California

residents please add applicable sales tax. Payments **must** be enclosed with the order and **must** be in US dollars payable on a US bank.

USENIX Conference and Workshop Proceedings

Meeting	Location	Date	Price	Overseas Mail		Source
				Air	Surface	
USENIX	Atlanta	Summer '86	\$25	\$25	\$5	USENIX
Graphics Workshop II	Monterey	December '85	\$ 3	\$ 7	\$5	USENIX
USENIX	Denver	Winter '86	\$20	\$25	\$5	USENIX
USENIX	Portland	Summer '85	\$25	\$25	\$5	USENIX
USENIX	Dallas	Winter '85	\$20	\$25	\$5	USENIX
Graphics Workshop I	Monterey	December '84	\$ 3	\$ 7	\$5	USENIX
USENIX	Salt Lake	Summer '84	\$25	\$25	\$5	USENIX
UniForum	Wash. DC	Winter '84	\$30	\$20		/usr/group

EUUG Publications

The following EUUG publications may be ordered from the USENIX Association office.

The EUUG Newsletter, which is published four times a year, is available for \$4 per copy or \$16

for a full-year subscription. The earliest issue available is Volume 3, Number 4 (Winter 1983).

The July 1983 edition of the EUUG Micros Catalog is available for \$8 per copy.

T-Shirts

There have been many requests for various USENIX t-shirts. There are still some 10th anniversary shirts available, but only in extra-large; the other shirts (blazoned with USENIX or with USENIX Atlanta '86) are available in various sizes.

Because the office is not geared up for retail trade, we will put all the shirts on sale in Washington in January.

Third Annual USENIX Computer Go Tournament

Peter Langston[†]

The third USENIX Computer Go Tournament took place June eleventh and twelfth in Atlanta, Georgia during the USENIX 1986 Summer Technical Conference & Exhibition. The tournament took the form of a double round-robin with each entrant playing every other entrant twice, (once with white and once with black).

Six programs were entered: *goanna*, *goo*, *nemesis*, *ogo*, *revjim*, and *scrappy*.

■ *Goanna* was written by Bruce Ellis (*research!brucee*), and is the only program that has been entered, unchanged, in all three USENIX Go tournaments. It employs a table-driven pattern-matching strategy that requires almost no cpu time to run (averaging 0.084 seconds per move).

■ *Goo*, written by Peter Langston (*{yquem,bellcore}!psl*), claims to be the oldest living descendent of *oog*, last year's champion; (*oog*'s absence from this year's tournament gave credence to rumors of foul play, intrigue, and missing backup tapes). Unlike other programs attributed to Mr. Langston (the tournament organizer), *goo* seemed quite earnest and innocent of guile.

■ *Nemesis* was written by Bruce Wilcox (*wilcox@bbng*) and is available commercially for a variety of micro- and mini-computers. As was the case last year, it entered the tournament as the clear favorite, having demolished the opposition in the first USENIX tournament and having played well in tournaments with humans. But once again, *nemesis* had trouble with the tense tournament atmosphere and after crashing in its first six games decided that

something had gone awry with the last minute port to the tournament machine and dropped out.

■ *Ogo* was written by Peter Langston (*psl@mouton*). It uses a peculiarly mindless strategy which involves playing the mirror image of each of its opponent's moves; (this symmetric strategy was in use as early as 854 A.D.). This strategy, while clever, has several drawbacks, the most serious being that, to avoid a stalemate, the program must eventually be able to play on its own. Fortunately, this year's *ogo* appears to have taken some lessons from last year's *oog*...

■ *Revjim* was written by Hank Dietz (*polyof!hank? polyof!dietz? c/o polyof!john?*). Spectators at the first USENIX Go Tournament will remember *jim*, the program that preferred death to dishonor and would choose suicide whenever in doubt (i.e. in every game). The fears that the *rev* in *revjim* stood for *Reverend* rather than *revised* were laid to rest when the program managed to avoid suicide this year. Unfortunately, *revjim* appeared to misunderstand the need for "two eyes" in every group and spent much of its time filling in its own groups until there were *just* two eyes left.

■ *Scrappy* was written by Herbert Enderton (*Herbert.Enderton@sam.cs.cmu.edu*) and was this year's surprise contender. Appropriately named, *Scrappy* favors a tactical approach, doing well in the close in-fighting, but losing ground for lack of large-scale "shape." Nonetheless, *scrappy* managed to win all but two of its games and those were only lost by slim margins. This will be a program to watch next year.

Third Annual USENIX Go Tournament
Game Scores

BLACK	goanna	goo	WHITE ogo	revjim	scrappy	W/L	points
goanna	---	-365	-20	-20	-368	0/4	-771
goo	206	---	42	185	6	4/0	437
ogo	351	-67	---	144	-27	2/2	401
revjim	*-	-143	-41	---	-195	0/3	-377
scrappy	225	-11	94	183	---	3/1	490
W/L	0/3	4/0	2/2	1/3	3/1		
points	-780	584	-75	-491	583		

[†] The author of "Results of the Second Annual USENIX Go Tournament" that appeared in the May/June 1986 issue of ;login: was J. P. Buhler. Our apologies for the error.

;login:

The problems that beset the judging last year were largely missing in this tournament. Only one program had trouble with illegal moves, and that program was withdrawn from the tournament; the remaining programs either succeeded or failed based on their relative abilities (or inabilities) to play Go. With the exception of one game that was judged "incomplete" (*revjim* vs. *goanna*), the judging simply consisted of determining the score at the end of each game and, once all the games were scored, ranking the entrants by the ratio of games won to games played.

Third Annual USENIX Go Tournament Final Standings				
Place	Entrant	Won/Lost	Ratio	points
1	goo	8/0	1.000	1021
2	scrappy	6/2	0.750	1073
3	ogo	4/4	0.500	326
4	revjim	1/6	0.143	-868
5	goanna	0/7	0.000	-1551

With *nemesis* out of the competition, it was no great surprise that *goo* won the tournament [breeding

shows?]. It was a surprise, however, that the newcomer *scrappy* beat everybody but *goo* and came very close to beating *goo* in both of their games. It was also a surprise that *ogo* did well enough to come in third, (implying substantial improvements in *ogo*'s non-mirror playing). *Revjim* appears to be on the road to recovery, having overcome its suicidal tendencies; with some accelerated therapy it may return as a real contender next year. *Goanna*, unchanged throughout the three USENIX tournaments, gives us a relative measure of the level of play in the three tournaments. In the first tournament *goanna* came in second, in the second tournament *goanna* slipped to third, in the third tournament *goanna* has fallen to last. Or, put another way; the same program that came in second the first year couldn't win a single game this year – the programs *are* improving.

Plans are being made for the Fourth Annual USENIX Go Tournament, to be held at the 1987 Summer USENIX Conference. Volunteers are being solicited for a small committee (3 or 4 people) to organize the tournament. If interested, contact *yquem!psl*, *bellcore!psl*, or *psl@mouton*.

START WORKING ON THOSE GO PROGRAMS!
YOU CAN DO BETTER THAN THESE GUYS, CAN'T
YOU?

The VAX and I

[Editor's note: This column is a slightly modified transcription of an introductory UNIX® talk given by Jin Mazumdar at the Department of Math and Computer Science at the State University of New York, College at Fredonia. -RBK]

A few weeks ago I was introduced by a colleague to a lady in another department. The introduction went something like this: "This is Jin. He takes care of our VAX."

The lady gave me a very sweet smile and replied by the way of conversation "How nice; how nice of the Math Department to have someone full time to look after their vacuum cleaners." Since that day I have taken care to explain terms like VAX and UNIX to everyone I met.

Let us take a very light hearted look at the VAX. I will try to show how the VAX to me is like a mother, wife, friend, teacher, secretary, analyst, mailman, and more all combined in one. Although I will be taking a rather frivolous look at the VAX I must mention at this point that everything I will discuss is currently a working product.

First, let's start the day with the problem of getting up from bed. Early rising has always been a problem with me and it always helps if there is someone to wake me up. The VAX does this by giving me a phone call at a prearranged time. Of course it is a little disconcerting to pick up the ringing phone in the wee hours of the morning (that is before noon) and find no one at the other end but one soon gets used to it. Unfortunately, at present the VAX will call you on phone but it will not talk to you. However, if I decide that I do not want to get up the VAX will hang up the phone after about 5 rings and let me sleep on.

After I manage to drag myself from the bed and make it to the department and sign on to the VAX, I have all my mail waiting for me on the VAX. We have a mail system on the VAX through which we can send or receive electronic mail from anywhere in this

;login:

country or even abroad. The VAX arranges to have all my mail sorted and each morning I get a display showing who each mail is from and the subject of each letter which I can then read at leisure.

Of course a lot of our intradepartmental correspondence is carried on via the VAX. I do believe that this fact is the key to there being such good relations amongst the faculty in the Math Department. Whenever we have something to say we always dash off a note to our colleagues on the VAX. So thanks to the VAX we do not actually have to go and talk to the other person. If you believe in the saying that familiarity breeds contempt you can figure out how the VAX is instrumental in the maintenance of good relations in the Math Department. Besides with this system of electronic mail we save trees as there is no paper involved.

At the same time the VAX also reminds me of the things I have to do that day or in the near future. This makes sure that I do not forget Mom's birthday or to send flowers to an old flame.

To deal with daily correspondence there are a number of editors available ranging from very simple ones to fairly complex ones. After one is done with drafting a letter or a note one can leave it up to the computer to find the mistakes. The computer just zips through your text and finds all the spelling mistakes you made. Of course you have the option of deciding whether you want your text to be checked for spellings the British way or the American. This feature of having the VAX go through one's composition and find out all the spelling mistakes may not be a big boost to one's ego but it is certainly a very handy feature to have.

Well if spelling correction is not too great for one's ego, one should not try to use another feature called *diction*. If one invokes this feature, the computer goes through the text and finds all the lengthy expressions and badly formed phrases and

suggests how you should rewrite your own text. On request the computer will also run your text through three readability tests and report your scores back to you.

During the course of work if I decide that I need a short diversion I can always ask the VAX to print a joke for me. Of course here too I have an option of specifying what type of jokes I would like to read. The infamous -o option to the command gives me an obscene joke. As our system was written at Berkeley and the jokes came along with it I have come to believe that the guys at Berkeley do have a good sense of humor, albeit somewhat perverted.

If through the day you want some change or you have been working through the night and need some relaxation there are a host of games available on the VAX. The games range from serious games like chess to a real trivial one called worms where all you see is three worms crawling over your screen.

With so many things one can do on the VAX one might wonder how one would go about doing all these. The VAX is an excellent teacher and has on-line tutorials which take you by the hand (or rather by your fingers) and patiently explain and try to teach you more about the various utilities. The VAX is a considerate teacher and every 20 minutes or so will ask you whether you would like a coffee break.

If at any time of the day one gets the feeling that one is going crazy with all this information and computing, one can always go for help to the on line analyst. The doctor program on the system will talk to you, ask you questions and serve you as your analyst. To top it all the doctor will also send you a bill by return mail. This doctor is perhaps the worst because he charges not by hour but by seconds of computer time used.

The VAX is our friend. I hope you enjoy him as much as I do.

4.3BSD Manuals

The 4.3 Berkeley Software Distribution became available at the end of May. A completely new set of manuals has been in preparation and should be ready soon after Labor Day. Where the 4.2 Manuals took up five volumes, the 4.3 manuals will be in seven volumes. The last volume will be a full index to the other volumes, prepared by Mark Seiden.

While the exact price is still uncertain, it will be about \$70 for the whole set. Nowadays this is a bargain price for books, to say nothing of technical manuals.

Further information, ordering requirements, and an order form will be available in the next issue of ;login:.

HoneyDanBer UUCP – Bringing UNIX® Systems into the Information Age

Bill Rieken
Middletown, NJ
ihnp4!opus!mtkam!wdr

Jim Webb
Bernardsville, NJ
ihnp4!opus!oliver!jrw

Part 2: Error Handling, Administrative Aids, and User Enhancements

Introduction

In Part 1[†] we described three major enhancements provided by HoneyDanBer UUCP: **spool** directory trees, flexible **permissions**, and a variety of **dialers**. In Part 2 we will illustrate other enhancements that make HoneyDanBer easier to use and take care of.

We start with a discussion of improvements in error handling. In the context of its environment (i.e., UNIX® systems) HoneyDanBer is far more “robust” than its predecessor. Comparisons of how errors were and are handled should demonstrate more reasons why HoneyDanBer UUCP does indeed help bring UNIX systems into the Information Age. You may even get a few ideas to use in systems you are currently developing.

Next we describe the administrative files and commands provided by HoneyDanBer UUCP for basic monitoring and control of the UUCP subsystem. We believe that you will agree that they are much better organized and easier to use than the administrative aids supplied with the previous version of UUCP.

Finally we describe a few of the user-level enhancements that make our UNIX work a little easier.

Error Handling

The word “robust” is usually expressed as wishful thinking when talking about UNIX systems. Part of the reason for this is the “silence is golden” philosophy not to bother programmers with superfluous messages. For example, the command line in Figure 1 makes the disk whir a bit, creates a file, and then stops. Without the **-t** option, **mm** did not invoke the **tbl** preprocessor, causing tables in the input file to appear as a mess in the output file. It could have **grepped** for table-formatting commands and invoked **tbl** automatically, or, at least printed a warning message! But why bother a busy programmer with minor details? After all, every option is well documented in the manual...

```
mm -rL60 -rW68 [0123456789].* >/tmp/mm.out 2>&1 &
```

Figure 1: Silence is Golden

Some UNIX utilities carry this a bit too far, and real errors are simply ignored. Consider the **mkfs** example in Figure 2. Notice how **mkfs** kept on writing every disk block in the 512-Megabyte file system without checking any of the **write** system calls!

Version 2 UUCP was equally as graceful when the spool file system ran out of space. Once it finished its writing loop, it went away, leaving a console full of “File System out of space” messages. This is one way a **mailed** file could get lost midstream, and neither end would be notified.

This article is © 1986 by Bill Rieken and Jim Webb – All Rights Reserved. It may be reproduced only by USENIX Association members for personal or in-house, non-commercial use. It may not be reproduced or distributed in any form for any other use without permission of the authors.

[†] “HoneyDanBer UUCP – Bringing UNIX Systems into the Information Age,” in the May/June 1986 issue of *;login:*, pages 27-36.

;login:

```
# mkfs /dev/dsk/c4d0s0 1000152:65536 1 1044
WARNING: AT&T-LAS FAULT: board 4, drive 0, slice 0
OPCODE=21, PHASE=5, STATUS=1, RTAC=1, BLOCK=4
    < 1,000,148 error messages >
WARNING: AT&T-LAS FAULT: board 4, drive 0, slice 0
OPCODE=21, PHASE=5, STATUS=1, RTAC=1, BLOCK=1000151
mkfs: Available blocks = 998100
#
```

Note: The 512-MB AT&T-LAS controller board was intentionally disconnected from the 3B2-400 to create this example.

Figure 2: **mkfs** Error Handling (?)

HoneyDanBer UUCP checks every I/O system call for successful completion, and initiates appropriate error handling when necessary. As one example, Figure 3 shows what you should expect whenever the receiving system does not have enough room in its spool directory to hold the file being sent to it. The **uutest** command will be discussed in detail later, but for now you should know that we used it just to *start* sending the files queued in our local ("mtkam") spool directory to the remote ("sirius") system. Recall from part 1 that UUCP notes any error conditions in a *status* file, and aborts the transmission. As you can see in Figure 3, HoneyDanBer keeps this information in */usr/spool/uucp/.Status/sirius*. Version 2 UUCP would have recorded it in a file called */usr/spool/uucp/STST.sirius*.

Try to start a file transfer...

```
$ uutest sirius
10 1 515553948 300 ASSERT ERROR
remote system can't create temp file sirius (uucico) sirius
Continue? n
```

Check the remote system's *status* file:

```
$ cat /usr/spool/uucp/.Status/sirius
10 1 515553948 300 ASSERT ERROR
remote system can't create temp file
```

Later, you will get mail...

```
From nuucp Sat May 3 21:42 EDT 1986
REQUEST: mtkam!D.siriu7036deb --> sirius!X.siriusN7036 (wdr)
(SYSTEM: sirius) remote system can't create temp file
```

Figure 3: HoneyDanBer File System Space Checking

However, as mentioned before, this error condition would have gone unnoticed in the older Version 2 UUCP. One small, tedious step for the HoneyDanBer programmer to code a **ustat(2)** system call to check for enough file system space *before* transmitting the file. A giant leap for *netnews* readers everywhere.

Cryptic, vague, and ambiguous error messages have been a (the?) hallmark of UNIX systems since the early days. How many times have you gotten the message, "foo: can't open file," and discovered yet another cause for that single message?

;login:

HoneyDanBer may set an example for "robustness" in UNIX programming. Figure 4 shows one example. Not only is the sender notified of the problem, but the error is clearly visible. Moreover, the standard input is returned to the user. In this case it saves the sender from re-running the monster **nrff** process on a 379-page book. In other cases it might return the only copy of valuable data.

```
$ nrff -cm memo | uux -b -p hru3d!lp -d2nd
    < time passes >
$ mail
From uucp Sun Feb  9 13:23 EST 1986 remote from hru3d
To: opus!jrw
remote execution [uucp job hru3dN1ae8 (2/9-13:23:13)]
    lp -d2nd
exited with status 1

    === stderr was ===
lp: destination "2nd" non-existent

    === stdin was ===
uucp is a standard networking package available on almost
all implementations of UNIX. It provides for the transfer
of files between both local and remote machines, as well as
allowing for the remote execution of commands ...
```

Figure 4: HoneyDanBer Error Message

Figure 5 shows another example of how HoneyDanBer improved error messages from Version 2 UUCP.

;login:

Ambiguous Version 2 UUCP Message:

```
$ uuto 9.toc opus!jrw
permission denied /u/wdr/hdb/9.toc
uucp failed partially: 1 error

$ ls -ld /u/wdr /u/wdr/hdb 9.toc
drwxr-xr-x 7 wdr A-team 768 May 27 04:00 /u/wdr
drwxr-xr-x 28 wdr A-team 592 May 25 08:57 /u/wdr/hdb
-rw-r--r-- 1 wdr A-team 10 Apr 20 18:09 9.toc
```

Unambiguous HoneyDanBer Message:

```
$ uuto 9.toc opus!jrw
bad system: opus
uucp failed completely (11)
```

Figure 5: Comparison of Error Messages

Actually, neither message tells you *how to fix the problem*, but at least the HoneyDanBer message tells you in no uncertain terms that your request failed. The Version 2 UUCP sends you out “chasing rainbows” in left field: What does “failed partially” mean? Did UUCP send *half* the file? Who was denied permission? Why? (All directories have “rwxr-xr-x” permissions, and the file is “rw-r--r--”). Did *anything* get sent?

The unfriendly HoneyDanBer message (“uucp failed completely”) was caused by improper file permissions. `/usr/bin/uucp` must be owned by `uucp` with the set-user-id bit on (i.e., “---s--x--x uucp”). Otherwise the user cannot read the *Systems* file and “uucp fails completely” when invoked from the `/usr/bin/uuto` shell script.

Maybe “IEFBR14SOC09” doesn’t make any sense, but at least you can look it up in a Big Blue error manual and find out what happened. Rather than document all errors in one place, UNIX systems expect that all **uuto** users also know about the **uustat** command. Figure 6 compares what **uustat** tells you in Version 2 and HoneyDanBer UUCP.

;login:

```
$ mail opus!wdr < to_bill
$ uustat
```

With Version 2 uucp...

1002	jrw	opus	02/01-16:22	02/01-16:22	JOB IS QUEUED
local			time of the job's		the status
user			submission		of the job
destination					
machine					

With HoneyDanBer uucp...

job id	time of job's			size of	the
	submission			the file	file
opusN66a0	02/01-16:22	S	opus	jrw	65
	02/01-16:22	S	opus	jrw	rmail
		S = sending		local	command
		R = receiving		user	
		destination			
		machine			

Figure 6: Comparison of **uustat**

It's debatable whether either of these outputs will help a casual UNIX user, but certainly HoneyDanBer gives the system administrator more and better information. For example, the administrator can find the *D.* file directly without searching for it, the job id is given, and the **rmail** command tells the administrator how the file was sent. Also, the local administrator can tell a remote administrator how much spool space is needed for the file transfer, in case manual intervention is needed. (e.g., recall the example shown in Figure 3.)

The new **uustat** also has more options than the previous version. The **"-p"** option helped us troubleshoot a recent problem we had. A **uuto** transfer of a very large **cpio** file was in progress when a power outage broke the DATAKIT connection on the sending side. The commands shown in Figure 7 were used to find out why the file had not arrived on the receiving system. Notice how the HoneyDanBer spool directory tree makes it easy to find what, if anything, has been received from a remote machine. In Version 2, we would have to hunt for the files we wanted among the conglomeration in */usr/spool/uucp*!

;login:

```
$ mail                      ! the next morning, we expected
No mail.                    ! notification from uuto

$ ls -l /usr/spool/uucppublic/receive/wdr/opus
total 0                      ! nothing from uuto!

$ ls -l /usr/spool/uucp/opus ! anything from opus?
total 1124
-rw----- 1 uucp other  570368 Mar 25 16:33 TM.14355.000
                        -----
                        ! yes, but nothing since 4:33 pm yesterday!

$ uustat -p                  ! what's going on?
LCK..opus: 14355             ! process 14355 holds opus lock

  UID  PID PPID  C PRI NI ADDR SZ  WCHAN   STIME  TTY    TIME  CMD
nuucp 14355    1  28  28  20 1281 24 283dc0 14:45:25 tty1 255:36 -uucico
  -----
                        ! uucico has been running since 2:45 pm yesterday,
                        ! chewing up 4+ hours of CPU time,
                        ! without moving a byte after 4:33 pm yesterday!
```

Figure 7: Output of `uustat -p`

The Develcon switch on the receiving side did not kill the `uucico` process when Carrier Detect dropped. As an experiment, we used the `uutest` shell script shown in Figure 8 to try to restart the file transfer.

;login:

```
# uutest --- test UUCP link to a remote machine
#
# usage:      $ uutest  SYSTEM  [ DEBUG LEVEL ]
#
# method:     determine if "new" HoneyDanBer UUCP or "old"
#              Version 2 UUCP by testing for /usr/lib/uucp/Systems
#              ("new" UUCP) or /usr/lib/uucp/L.sys ("old" UUCP).
#              Check for status file, show it to user, and ask
#              if user wants to continue.
#              If continue, then remove status file, set debug
#              level, and start uucico in the background.
#              Then 'tail' its output. Note: tail is needed so
#              that you can use break/delete key to stop it;
#              'uucico' ignores break/delete key.
#-----
# determine which version of UUCP this system uses
if [ -f /usr/lib/uucp/Systems ]
then
    STATUS=/usr/spool/uucp/.Status/$1
else
    STATUS=/usr/spool/uucp/STST.$1
fi
# look at the status file, if any ...
if [ -f $STATUS ]
then
    cat $STATUS
    echo "Continue? \c"
    read CONTINUE
    if [ "$CONTINUE" = "y" -o "$CONTINUE" = "yes" ]
    then
        rm -f $STATUS
    else
        exit
    fi
fi
# see if user specified a debug level
if [ "$2" ]
then
    LEVEL=$2
else
    LEVEL=5
fi
# now call the remote system
/usr/lib/uucp/uucico -r1 -x$LEVEL -s$1 >/tmp/$1.$$ 2>&1 &
# and watch the dialing/login sequence as it happens
tail +1f /tmp/$1.$$
#-----THE END
```

Figure 8: A Local **uutest** Shell Script

;login:

Yes, we know that HoneyDanBer provides a similar **Uutry** shell script. Our shell script is not “yet another way to write a Uutry command.” It serves three purposes for our convenience:

1. **uutest** works with both versions of UUCP
2. **uutest** displays any *status* file, and removes it if we want to proceed
3. **uutest** is easier to type than **Uutry**

Since we are responsible for nearly twenty different machines, and not all of them have HoneyDanBer UUCP, item one is very handy for us. The **uucico** command is programmed to stop if it finds a status file, so item two saves us the aggravation of being interrupted in our troubleshooting to go do the “busy work” of removing the status file. Much like item two saves us the extra “-r” keystrokes required by **Uutry**, the third item relieves us of the minor nuisance of typing an uppercase letter. Yes, we know that the “debug level” option could have been programmed in the **shell** as “LEVEL=\${2:-5}”, but we try to make our shell scripts readable for others, and, besides, this is just a magazine article, not a famous book.

Getting back to our example, the **uutest** output in Figure 9 shows that HoneyDanBer UUCP looked for an **opus** lock file, saw that the process holding the lock file was still running, and disconnected. Version 2 UUCP also looked for lock files to prevent two simultaneous conversations between the same machines, but it never checked to see if the process holding the lock file was still running! If either of two talking machines crashed, it is possible that the lock file would stay in the file system, while the process using the shared resource disappeared when the system went down. Another “busy-work” task for system administrators to check after re-booting from a system crash.

```
$ uutest opus
6 1 513090623 300 LOGIN FAILED opus
Continue? y
ulockf file /usr/spool/uucp/LCK..opus
Lock File--process still active--not removed
Currently Talking With opus
SC= in cleanup... code=100
call undial(-1)
Conversation Complete: Status FAILED
```

Figure 9: **uutest** of Lock File Handling

HoneyDanBer eliminates this chore by handling lock files in a more intelligent way. When we killed the nuucp process (14355) on the receiving system, and started **uucico** again on the calling system, HoneyDanBer looked for the process id in the lock file, saw it was not running, replaced the contents of the lock file with the new **uucico** process id, and started a new connection.

Version 2 UUCP would not connect these two machines again until someone manually removed the lock file, or a two hour time limit had expired. It was this arbitrary time limit that caused the more serious problem: if a second request arrived during a *long* data transfer like the one shown in Figure 7, old UUCP would assume that the lock file was “too old,” and it would give the shared resource (e.g., a modem port) to the second requestor, too!

The **uutest** (**Uutry**) command is probably the most useful tool you have to troubleshoot UUCP problems. It is impossible to show all the problems you may encounter with various communications devices, but it is quite easy to remember that wherever **uucico** stops – *that’s* where your problem is! We’ll see a few more examples later, but for openers let’s compare **uutest** output from Version 2 UUCP with that from HoneyDanBer.

;login:

Figure 10 shows how Version 2 UUCP dials another machine, and Figure 11. shows how HoneyDanBer does it. You can decide for yourself which one is easier to follow. One reviewer commented, "So what? One *wants* cooperation, and the other *expects* cooperation. They're both looking for a tie-on (TION:). It's all Greek to me!" We agree. The usefulness of this tool is that it shows you where **uucico** stops.

```
$ utest opus
4 1 512596209 3300 LOGIN FAILED opus
Continue? y
finds called
getto called
call: no. 5551212 for sys opus call phone number 5551212
login called
wanted "" got that
NO NL
wanted TION: <0xd><0xa>ORIGIN: DATAKIT VCS NODE a1
MODULE 96 PORT 1<0xd><0xa><0xd><0xa>DESTINATION:got that
wanted in: opus<0xa>got ?
wanted in: <0xa><0xd><0xa>opus/1200<0xa><0xd>login:got that
wanted word: nuucp<0xa><0xd>Password:got that
valid sys Shere=opus          =====
                                (note: not printed!)
msg-ROK
  Rmtname opus, Role MASTER, Ifn - 5, Loginuser - wdr
rmsg - 'P' got Pg
wmsg 'U'g
Proto started g
protocol g
*** TOP *** - role=1, setline - X
wmsg 'H'
rmsg - 'H' got HY
  PROCESS: msg - HY
HUP:
wmsg 'H'Y
cntrl - 0
send 00 0,exit code 0
```

Figure 10: **utest** with Version 2 UUCP

;login:

```
$ grep ph1 /usr/lib/uucp/Devices
ACU ph1 ph1 1200 PC7300
|
+---- note column 1 ----+
|
$ uutest opus
1 1 512604766 300 NO DEVICES AVAILABLE opus
Continue? y
conn(opus)
Device Type ACU wanted
Using pc7300 caller (dev=ph1)
Use Port /dev/ph1, Phone Number 5551212<
getto ret 5
expect: ("" )
got it
sendthem (<NO CR>)
expect: (TION:)
sendthem (^M)
expect: (TION:)
^M^JORIGIN: DATAKIT VCS NODE a1 MODULE 96 PORT 1
^M^J^M^JDESTINATION:got it
sendthem (^M)
expect: (in:)
opus^M^Jsendthem (^M)
expect: (in:)
^M^M^Jopus/1200^M^Jlogin:got it
sendthem (^M)
expect: (word:)
nuucp^M^JPassword:got it
sendthem (^M)
Login Successful: System=opus
```

Figure 11: **uutest** with HoneyDanBer UUCP

Remember that HoneyDanBer uses a *Dialers* file to separate the device-dependent knowledge from the rest of the system. However, note that each *Devices* file entry *must begin in column 1*, as indicated in Figure 11. Otherwise you will get a "NO DEVICES AVAILABLE" message. The column 1 restriction is not mentioned in the manual. However, at least the HoneyDanBer error manual points you in the right direction, as shown in Figure 12. That's more than Version 2 UUCP ever did for you!

;login:

NO DEVICES AVAILABLE

There is currently no device available for the call.

(We could have guessed this from the message.)

Check to see that there is a valid device in the *Devices* file for the particular system.

(We did, and there was.)

Check the *Systems* file for the device to be used to call the system.

(We did, and it was there.)

Figure 12: HoneyDanBer Error Manual Example

Before we close this section on error handling, we'd like to show you a *few* of the error indicators you may get when you use the **uutest** command. Some of the common ones are shown in Figure 13. While the HoneyDanBer UUCP ("Basic Networking Utilities") manual is far better than anything published prior to it, there is no substitute for actually watching the **uucico** actions on your terminal screen as they happen!

uucico always tries twice:

```
^M^JDIALING: 17075551212^M^JNO RING^M^J>timed out
```

```
Connect failed: CALLER SCRIPT FAILED
```

```
^M^JDIALING: 17075551212^M^JNO CD^M^J>timed out
```

```
Connect failed: CALLER SCRIPT FAILED
```

... **Systems** script is fine! Their modem isn't answering!

New **cu** has a "-d" option:

```
$ cu -d 17075551212
```

```
Device Type ACU wanted
```

```
Connect failed: DEVICE LOCKED -----+
```

```
call cleanup(1)
```

```
call _mode(0)
```

```
$ ls -l /usr/spool/locks
```

```
total 1
```

```
-r--r--r--  1 uucp      5   11 Apr 29 17:28 LCK..tty110
```

```
$ file /usr/spool/locks/*
```

```
/usr/spool/locks/LCK..tty110:  ascii text
```

```
$ cat /usr/spool/locks/*
```

```
3982
```

```
$ ps -fp 3982          ! who's using the port?
```

```
  UID  PID  PPID  C   STIME TTY      TIME COMMAND
stewart 3982  2064  6  17:28:02 tty121  0:05 cu -ltty110 -s1200
```

;login:

```
$ ps -fu stewart          ! what are they doing with it?
  UID    PID  PPID  C   STIME  TTY      TIME  COMMAND
stewart  2064    1    0 16:07:10 tty121    0:05  -sh
stewart  3982  2064    0 17:28:02 tty121    0:05  cu -l tty110 -s1200
stewart  3983  3982   39 17:28:02 tty121    1:22  cu -l tty110 -s1200
```

... Note: one cu 'sends,' the other one 'receives'

```
From uucp Wed Apr 30 08:26 EDT 1986
>From uucp Wed Apr 30 07:21 EDT 1986 remote from opus
>From uucp Wed Apr 30 07:19 EDT 1986 remote from attunix
remote execution [uucp job attunixN5479 (4/30-7:19:54)]
      uucp -C -nwdr bar.c tarpon!~/receive/wdr/mtkam//
execution permission denied to nuucp
```

... 'tarpon' does not have "COMMANDS=uucp" in *Permissions*

Figure 13: HoneyDanBer UUCP Troubleshooting

Administrative Aids

The UUCP subsystem was perhaps the most troublesome area with the most vexing problems for UNIX system administrators, prior to the HoneyDanBer enhancements. The variety of modems, each with its own peculiarities, and the number of hardware connections, each with a possibility of causing a break in the data communications circuit, made UUCP troubleshooting much more difficult than other subsystems such as LP spooling, system accounting, or RJE links to an IBM mainframe. The large number of nodes in the distributed *Systems* file increased the probability that some sites either changed their node name, their *uucp* login password, their telephone number, or added a local area telephone switch which changed the *expect-send* login sequence to each site on the local switch.

These kinds of problems are much easier to identify and remedy using HoneyDanBer UUCP. You have already seen some of the troubleshooting tools in the previous section. In this section we will show you a "UUCP Roadmap" of the administrative files and commands for basic monitoring and control of the UUCP subsystem.

Figure 14 shows the new names of UUCP administrative files. Probably the best way to find out if a system is running HoneyDanBer is to check for *L.sys* (old UUCP) or *Systems* (HoneyDanBer) in the */usr/lib/uucp* directory. *L-devices* is now two files, *Devices* and *Dialers*, to support a variety of new data communications devices. The awkward *USERFILE*, *ORIGFILE*, *FWDFILE*, and *L.cmds* files have been replaced by the simpler *Permissions* file. The *Permissions* file was described in the *Security Features* section of Part 1, and the *Dialers* file was described in the *Networking Facilities* section of Part 1.

;login:

UUCP Administrative Files

<u>Old UUCP</u>	<u>HoneyDanBer</u>
L.sys	Systems
L-devices	Devices Dialers
L-dialcodes	Dialcodes
USERFILE ORIGFILE FWDFILE L.cmds	Permissions

Figure 14: Administrative Files in */usr/lib/uucp*

While static control files and commands are kept in the */usr/lib/uucp* directory, active job queues and log files are kept in the */usr/spool/uucp* directory. As described in the *Performance Enhancements* section of Part 1, the */usr/spool/uucp* directory has been subdivided into separate directories for each remote site. Figure 15 shows the new directory structure in HoneyDanBer UUCP.

HoneyDanBer UUCP Directory Structure

	.Admin	Contains administrative files
	.Corrupt	Corrupt files are moved into this directory
	.Log	Log files are in subdirectories here
	.Old	Old log data
	.Sequence	System sequence numbers
	.Status	System status information
	.Workspace	UUCP temporary workspace area
	.Xqtdir	Remote executions take place here
<i>/usr/spool/uucp/</i>	oliver	Directories containing files to/from the specific systems
	mtkam	
	hru3c	
	hru3d	
	ihnp4	
	uucico	Directory of uucico execution logs
	uucp	Directory of uucp request logs
<i>/usr/spool/uucp/.Log/</i>	uux	Directory of uux request logs
	uuxqt	Directory of uuxqt request logs, or remote command executions run on the local machine

Figure 15: Administrative Files in */usr/spool/uucp*

;login:

Subdirectories simplify the implementation of commands like **uulog** shown in Figure 16. The "**uustat -m**" command shows the user how well UUCP is communicating with other systems. We notice a problem with *sirius* so we execute "**uulog -f sirius**" to find out how long this problem has been bothering us. The output shown in Figure 16 tells us that UUCP is having trouble logging in to *sirius*. It also implies that UUCP tries twice to reach a remote site, then waits a few minutes and tries again. Each failure to connect increases the waiting time before another attempt is made. After approximately one week, a program called **uucleanup** warns the sender of the problem a few days before the spool files are removed.

```
$ uustat -m
mtuxn          04/28-08:23 SUCCESSFUL
oliver         04/29-00:29 SUCCESSFUL
opus           04/29-07:18 SUCCESSFUL
ph0            Locked
pyuxc          04/27-08:16 SUCCESSFUL
pyuxqq         04/25-21:35 SUCCESSFUL
rruxqq         04/29-12:46 SUCCESSFUL
sirius         3C(1)    04/29-07:32 DIAL FAILED Retry: 0:03
tarpon         1C      04/30-08:29 DIAL FAILED
usenix         1C      04/30-08:44 Locked TALKING

$ uproc        ! local shell script to print user processes
wdr 6674      1 3 Apr 29 w1 0:12 ua
wdr 8535 6674 3 Apr 30 w2 0:26 ksh
uucp 11690    1 3 08:21:05 w5 0:00 uusched
uucp 11743 11690 3 08:43:11 w5 0:02 uucico

$ uustat -q
ph0            Locked
sirius         3C(1)    04/29-07:32 DIAL FAILED Retry: 0:03

$ uulog -f sirius
uucp sirius (4/28-20:24:11,7337,0) CONN FAILED (CALLER SCRIPT FAILED)
uucp sirius (4/29-7:21:21,4099,0) LOCKED (call to sirius )
uucp sirius (4/29-7:22:14,4116,0) LOCKED (call to sirius )
uucp sirius (4/29-7:22:51,4100,0) FAILED (LOGIN FAILED)
uucp sirius (4/29-7:24:16,4100,0) FAILED (LOGIN FAILED)
uucp sirius (4/29-7:24:16,4100,0) CONN FAILED (CALLER SCRIPT FAILED)
uucp sirius (4/29-7:27:47,4151,0) NO CALL (RETRY TIME NOT REACHED)
uucp sirius (4/29-7:27:47,4151,0) CAN NOT CALL (SYSTEM STATUS)
uucp sirius (4/29-7:32:06,4165,0) FAILED (LOGIN FAILED)
uucp sirius (4/29-7:32:39,4165,0) CONN FAILED (DIAL FAILED)
```

Figure 16: HoneyDanBer UUCP Administrative Commands

When your system is having trouble reaching a remote site, the **uutest** shell script given in the previous section can be used to find out the exact cause of the communication problem. This enabled us to see that the *sirius* modem would answer the telephone, but the computer would not issue a login prompt, causing the error messages shown in Figure 16. We also were able to determine that *tarpon* was down for "unscheduled hardware maintenance" (i.e., a system crash) and so it was not answering our telephone calls. Meanwhile, *usenix* was truckin' right along, receiving part 1 of this paper.

Figure 17 shows what will happen if you are lucky when you try **uutest** to connect to a remote system: the call goes through, login is successful, and the file transfers take place. The dialing progress reports are omitted from Figure 17 because they were shown in the previous section.

;login:

```
$ uutest opus          ! try to "kick-start" UUCP
... DIAL FAILED        ! from previous attempt
Continue? y
conn(opus)
Device Type ACU wanted
Using pc7300 caller (dev=ph1)
Use Port /dev/ph1, Phone Number 5551212<

< connection protocol progress reports... >

Login Successful: System=opus
wmesg 'U'g
Proto started g
Request: mtkam!/u/wdr/aim.bug1 --> opus!~/receive/jrw/mtkam// (wdr)
wmesg 'S' /u/wdr/aim.bug1 ~/receive/jrw/mtkam// wdr -dcn D.0 644 jrw
Request: mtkam!/u/wdr/aim.bug2 --> opus!~/receive/jrw/mtkam// (wdr)
wmesg 'S' /u/wdr/aim.bug2 ~/receive/jrw/mtkam// wdr -dcn D.0 644 jrw
Request: mtkam!/u/wdr/aim.bug3 --> opus!~/receive/jrw/mtkam// (wdr)
wmesg 'S' /u/wdr/aim.bug3 ~/receive/jrw/mtkam// wdr -dcn D.0 644 jrw
wmesg 'H'
-----
Remote Requested: opus!D.opus88a43e2 --> mtkam!D.opus88a43e2 (root)
wmesg 'S'Y
wmesg 'C'Y
Remote Requested: opus!D.mtkam23133e2 --> mtkam!X.mtkamN2313 (root)
wmesg 'S'Y
wmesg 'C'Y
Remote Requested: opus!D.opus88a61e9 --> mtkam!D.opus88a61e9 (root)
wmesg 'S'Y
wmesg 'C'Y
Remote Requested: opus!D.mtkam2314022 --> mtkam!X.mtkamN2314 (root)
wmesg 'S'Y
wmesg 'C'Y
-----
wmesg 'H'Y
send 00 0,SC= in cleanup... code=0
call undial(5)
in putgetty, fgetty=0x200
Conversation Complete: Status SUCCEEDED
```

Figure 17: **uucico -r1** "kick-start"

After the "Login Successful" message appears, we see that the calling system (*mtkam*) requests the called system (*opus*) to Use the general protocol "g". This is a modified X.25 packet protocol (using 64-byte packets) which generally ensures reliable data transfer of ASCII and/or binary files. Other protocols provided in the HoneyDanBer package are full X.25 (*xio.c*), DATAKIT (*dio.c*), and "error-free" Ethernet (*eio.c*). Unfortunately, the default protocol must be chosen system-wide, as it is a **#defined** constant in *parms.h*. However, protocol function calls can be added to *callers.c* as required by the associated data communications equipment. This requires source code modifications, but it is certainly an improvement to the older Version 2 UUCP software, which only provided the general (*gio.c*) protocol.

Next, the 'S' commands are used to Send queued files from *mtkam* to *opus*. You can watch the progress as each file is sent, in case there is a problem with one of the file transfers, or you can use the break/delete key to exit from the **uutest** command, once you are satisfied that the file transfers will finish successfully.

When *mtkam* finishes sending all its spool files queued for *opus*, it sends an 'H' command to request *opus* to Hangup the line. However, *opus* starts sending "Remote Requests" for **rmail** execution on *mtkam* to receive files mailed to *mtkam!wdr* from *opus*. If *opus* did not have any files queued for *mtkam* it would have agreed

;login:

immediately to hangup the line by sending the "Y" response you see in Figure 17. Finally, a "SUCCESSFUL" message is written to the `/usr/spool/uucp/.Status/opus` file. (Note: the `undial()` function is peculiar to the UNIX PC7300. Note also: the file transfers from *opus* to *mtkam* depend on correct settings of `SENDFILES=yes` and `REQUEST=yes` in the systems' *Permissions* files, as described in the *Security Features* section of Part 1.)

Figure 18 shows the UUCP administrative daemons started by the system `cron` process. `uudemon.hour` is started at 21 minutes after the hour, every hour of every day. It basically tries to restart any queued file transfers waiting in the spool area. It could be run more often by changing the "21" to "1,21,41", for example, if your system has a lot of UUCP traffic. On the other hand, it could be run every four hours or so, if your system has little UUCP traffic. A nice feature of HoneyDanBer is that `uudemon.hour` invokes a new process called `uusched` to search the spool directories for file transfers, and `uuxqt` to process any outstanding remote executions on your system. This used to be done by `uucico` in old Version 2 UUCP, and the separation of the scheduling function improves the readability and performance of `uucico` in the HoneyDanBer package.

`uudemon.admin` is run daily at 4:30 am. It mails status information to the UUCP administrator and can also be run more frequently if desired. The output of "`uustat -p`" and "`uustat -q`" is automatically sent to the UUCP administrator in hopes that UUCP problems will be corrected before any user's work is hindered by UUCP failures.

`uudemon.cleanu` basically cleans up `/usr/spool/uucp/.Log` files by moving them to `/usr/spool/uucp/.Old`, and makes sure that files no longer needed are removed from the spool directories. It also mails a summary status report to the UUCP administrator at the end of its processing cycle. It, too, can be run more or less frequently as needed by your system. In the example shown in Figure 18 it is run once a week, at 5:30 am on Monday mornings. It is a very smart program with many heuristics to decide how to handle "orphaned" files (i.e., *C*, *D*, and *X* files that somehow lost their partners).

One morning we happened to notice *two* copies of `uudemon.cleanu` were running! This was noticed after we upgraded a UNIX PC7300 from System V Version 2 to Version 3. (Note: The version number indicates which one of Convergent Technologies' System V Release 1 UNIX ports is installed on the Motorola 68010-based UNIX PC7300. It does not have any connection with AT&T's System V Release 2 or the new AT&T Release 3.) Convergent Technologies Version 3 has a built-in `cron` function, whereas Version 2 started a `cron` process in `/etc/rc`. We forgot to change `/etc/rc` after the upgrade to Version 3, and that is why `uudemon.cleanu` was started twice that day. The fact that neither copy got confused by the other is a credit to the "robustness" of HoneyDanBer software.

Figure 18 also shows how HoneyDanBer processes `uuto` requests. There is *one* *C* control file in the `/usr/spool/uucp/sirius` directory, which tells UUCP to send several files to *sirius* whenever a connection is made. Because the files have "`-rw-r--r--`" permissions `uucp` will send directly from the named files and not waste spool space with duplicate copies. On the other hand, if the files had "`-rw-----`" permissions (i.e., owner access only), `uucp` is smart enough to make `uucp`-owned copies in the spool area at the time it is running with owner access permissions, so that `uucico` will later have access to the file. `uuto` invokes `uucp` with the "`-C`" option only when it, itself, was invoked with a "`-p`" option. Old Version 2 `uucp` was not as smart because requests to transfer "owner access only" files would result in an "access permission denied" error.

There is one more administrative daemon, `uudemon.poll`, which can be used to periodically call the remote machines listed in `/usr/lib/uucp/Poll`. Essentially all this shell script does is create dummy *C* files for the remote machines so that `uudemon.hour` will arrange to have these systems called. Obviously `uudemon.poll` should be scheduled to be run by `cron` a few minutes before `uudemon.hour` runs.

;login:

```
$ grep uu /usr/lib/crontab          ! which UUCP commands are run by 'cron'?
30 4 * * * /bin/su uucpadmin -c "/usr/lib/uucp/uudemon.admin > /dev/null"
30 5 * * 1 /bin/su uucpadmin -c "/usr/lib/uucp/uudemon.cleau > /dev/null"
21 * * * * /bin/su uucpadmin -c "/usr/lib/uucp/uudemon.hour > /dev/null"

$ date
Mon Apr 28 05:32:10 EDT 1986          ! 'uudemon.cleau' has been started

$ ps -fu uucp                        ! what is it doing??
  UID   PID  PPID  C   STIME  TTY   TIME COMMAND
uucp  6409  6406   3 05:30:01 w1    0:00 sh          ! external 'cron' process
uucp  6417  6409   3 05:30:04 w1    0:02 sh
uucp  6503  6417   3 05:30:51 w1    0:00 mail
uucp  6502  6417  43 05:30:51 w1    0:00 sh
uucp  6555  6502  81 05:31:27 w1    0:00 find
uucp  6416  6412   3 05:30:01 w5    0:00 sh          ! built-in 'cron' process
uucp  6421  6416   3 05:30:04 w5    0:02 sh
uucp  6506  6421   3 05:30:51 w5    0:00 mail
uucp  6501  6421  72 05:30:51 w5    0:00 sh
uucp  6561  6501  71 05:31:31 w5    0:00 sh
root   6406    1   3 05:30:00 w1    0:00 sh          ! added to show parents
root   6412    1   3 05:30:01 w5    0:00 sh
root    43    1   3 Apr 27   w1    0:52 cron        ! to distinguish 'cron's

$ du -s /usr/spool/uucp              ! what's in the spool area?
127                                  ! 127 disk blocks total

$ du -s /usr/spool/uucp/sirius
3                                    ! only 3 blocks for sirius!

$ cat /usr/spool/uucp/sirius/C.siriusN7034
S /u/wdr/hdb/1.1.intro ~/receive/beccat/mtkam// root -dcn D.0 644 beccat
S /u/wdr/hdb/1.2.spool ~/receive/beccat/mtkam// root -dcn D.0 644 beccat
S /u/wdr/hdb/1.3.secure ~/receive/beccat/mtkam// root -dcn D.0 644 beccat
S /u/wdr/hdb/1.4.dial ~/receive/beccat/mtkam// root -dcn D.0 644 beccat
S /u/wdr/hdb/1.5.sumary ~/receive/beccat/mtkam// root -dcn D.0 644 beccat
```

Figure 18: HoneyDanBer Daemon Shells

Some typical mail messages from the **uudaemon** shells are shown in Figure 19. The first one from June 30 alerts the UUCP administrator to a possible security threat by an outside intruder. The second one from July 7 indicates a possible security threat by an inside intruder. The third message from March 30 shows how programming errors are handled intelligently at the time, and later reported by **uucleanup**. The message tells the administrator that process 206 was running at six pm on March 30 when it encountered an error at line 88 in the source program *gtcfile.c*, Version 2, release 1.

;login:

```
From uucp Mon Jun 30 05:31 EDT 1986
uucleanup ran:
du -s for spool is: 288      /usr/spool/uucp

Subject: uucp DENIED
nuucp oliver (6/23-18:26:06,25528,0) REMOTE REQUESTED
(mtkam!/usr/bin/less --> oliver!/usr/spool/uucppublic/receive/jrw/mtkam
jrw oliver (6/23-18:26:07,25528,0) PERMISSION (DENIED)

From uucp Mon Jul 7 05:31 EDT 1986
Subject: cleanup
usenix (7/7-5:31:00,2369,0) oprocess: unlink(usenix/A.usenixN5296) (0)

Subject: uucp DENIED
wdr tarpon tarponN1dc9 (7/3-6:15:03,4759,0) REQUEST
(mtkam!/u/wdr/net.manners --> tarpon!~/receive/wdr/mtkam
wdr tarpon tarponN1dc9 (7/3-6:15:03,4759,0) ACCESS (DENIED)

From uucp Mon Mar 31 05:31 EST 1986
Subject: uucp Admin
ASSERT ERROR (uucp) pid: 206 (3/30-17:59:41)
CAN'T LINK /usr/spool/uucp/opus/C.opusN73e1 (13)
[SCCSID: @(#)uucp:gtcfile.c 2.1, FILE: gtcfile.c, LINE: 88]
```

Figure 19: HoneyDanBer Daemon Mail

If you have a desk-top microcomputer, installing HoneyDanBer UUCP is as easy as loading floppy diskettes in the order given by the "Install Software Package" menu directions. We'll see an example in the next section. On a larger machine, installation is just about as easy. After loading the distribution tape onto disk, one edits the parameter file (*parms.h*) and types "make install". There is a very well-written *README* file to help you, and a new program called **uucheck** verifies your installation. **uucheck** is also useful after installation to interpret your *Permissions* file into very descriptive English (i.e., system so-and-so can do such-and-such and is restricted from this-and-that).

HoneyDanBer UUCP can be configured for any UNIX system and is backward compatible with Version 2. It is a standard part of the System V Release 3 distribution, with enhancements that take advantage of STREAMIO, Remote File Sharing, and Shared Libraries.

Initial Setup

It has been said that your (generic) first attempt to setup a UUCP configuration will take a couple of days. After that, it takes about two hours to setup your next system(s). In this section, we will try to save you time if you have never installed UUCP before. A typical installation on an AT&T 3B2-400 microcomputer is shown, and tested by making a UUCP connection to an AT&T UNIX PC7300.

First, you need to load the software. Both the 3B2 and the 7300 have nice, "user-friendly" menu screens to help you install a software package. On the 3B2, you type "**sysadm softwaremgmt**" and select *installpkg* to install the HoneyDanBer UUCP (*Basic Networking Utilities*) package onto the internal hard disk. You will be given instructions to load the (single) floppy diskette, and asked to press the return key when it has been inserted into the floppy disk drive. This is the easy part. An "Install" shell script is copied from the floppy into */tmp* and executed. It automatically reads the rest of the files on the floppy diskette, copies them into the proper directories (e.g., */usr/spool/uucp*, */usr/lib/uucp*, and */usr/bin*) and gives them the correct ownership and file permissions.

;login:

Second, you need to configure your UUCP system. The “user-friendly” “**sysadm uucpmgmt**” command makes this a very easy process, too. Figure 20 shows the sequence we used. Selection 2 (*pollmgmt*) is not required in our configuration.

sysadm uucpmgmt

```
1  devicemgmt          (add)
   enter port name: tty14  (the physical I/O port)
   select device type: hayes (the ACU modem)
```

Selection 1 adds the following two entries into */usr/lib/uucp/Devices*:

```
ACU  tty14 - 1200 hayes
ACU  tty14 - 300  hayes
```

```
3  portmgmt           (modify tty14)
   bidirectional       (wishful thinking on a Hayes modem)
   speed: 1200
```

Selection 3 modifies the “14:2” entry in */etc/inittab* to:

```
14:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty14 1200H
```

But you might as well leave it “*/etc/getty -t 60 tty14 1200H*” for a Hayes, and alternate between *respawn* (login) and *off* (dialout), because switch 6 (Carrier Detect) is not programmable – it must be changed manually!

```
4  systemmgmt         (add)
   nodename: mtkam
   device to call on: acu   (automatic calling unit - a modem)
   speed: 1200
   phone number or switch token: 5551212
   remote equipment dialing into: dialup
   loginid: nuucp
   password:              (why bother? uucico login shell,
                           and NOSTRANGERS protection,
                           besides the unlisted phone #)
```

Selection 4 adds the following entry into */usr/lib/uucp/Systems*:

```
mtkam ACU 1200 5551212 in:--in: nuucp
```

Figure 20: Setting Up UUCP on an AT&T 3B2 Computer

The files modified by each selection are shown in Figure 20 for readers who do not have a menu-driven “simple administration” user interface like **sysadm** on the 3B2, or the User Agent (**ua**) on the 7300. As you can see, the actual file modifications are simple, but the menus help you remember what needs to be done, and also help prevent silly typing mistakes that cause so much trouble in UNIX! Notice that the *Dialers* file does not need to be modified, since it already contains device-dependent chat scripts for Hayes, 801, direct, Develcon, Micom, Penril, Rixon, and Ventel equipment. Later, we’ll see an example where we add a DATAKIT script.

The third step is to *test* your UUCP setup. Fortunately, the “**cu -d**” command shown in Figure 13 and Figure 24 gives us a handy tool to check our *mtkam* connection. The first message we saw was “CAN’T ACCESS DEVICE” – HoneyDanBer’s equivalent to the “foo: can’t open file” message. Knowing that our *Systems* and *Devices* files had been set up properly by the menu interface, we decided to test our hardware connections.

First, we turned **getty** off by changing **respawn** to *off* in */etc/inittab*, and killing the process currently running on *tty14*. We noticed the TR (Data Terminal Ready) light go off on the Hayes front panel. Then we started a **getty** process on *tty14* by editing */etc/inittab*, followed by the command “**init q**”. We noticed the TR light came back on! A good sign that the computer is reaching the modem just fine.

;login:

The “**cu -d mtkam**” debugging output showed a *timeout* just before the “CAN'T ACCESS DEVICE” message. This indicates that the device (i.e., the Hayes modem) is not responding as it should, so we removed the front panel to check the switch settings. Sure enough, switch 6 was up! Recall from the *Networking Facilities* section of part 1:

Hayes Modem Switch 6:

Down & Out (Carrier Detect always on)

Up & In (Carrier Detect recognized)

When we flipped switch 6 down, **cu** was able to *open* the port and dial out!

Once you get past the “CAN'T ACCESS DEVICE” hurdle, it's Easy Street from then on. Just follow the dialing/login sequence as it appears on your terminal screen, and you'll know where to fix any problems along the way. For example, we had to reboot *mtkam* because its login port was not answering our telephone call. We chose to reboot and do something else for awhile, since the port seldom gets hung.

After we logged in via the **cu** command, we were able to *%take* the *Systems*, *Devices*, and *Dialers* files from the 7300 to use on the 3B2. This required *super-user* privileges to read the *Systems* file, and we had to verify good transmission ourselves, since **cu** file transfers are not guaranteed to filter out line noise, etc. Happily our transfers were error-free, and our new 3B2 is up and running HoneyDanBer UUCP!

Finally, we'd like to show you a DATAKIT script to allow remote access to a local area DATAKIT network. You can use this as a template if you need to access ISN, Develcon, or Micom networks from a different building:

```
Systems:  tarpon Any DK 1200 9=5557777 in:--in: nuucp word: secret
Devices:  (3B2) DK tty14 tty14 1200 hayes 5551212 datakit dial penril \D
Devices:  (7300) DK ph1 ph1 1200 PC7300 5551212 datakit dial penril \D
Dialers:  datakit "" "" \d\r\c TION:--TION: \D
```

Figure 21: Device Pairs and a DATAKIT Dialer

You can read the manual for details, but basically we wanted to show you how easy it is to add a script to the *Dialers* file if the script you need is not included in the standard distribution. Also, the *Devices* file can have entries that define more than one device. In our example *tarpon* is reached as follows: DK is associated with an outgoing dial port (tty14 on our 3B2; ph1 on our 7300). The 3B2 uses the external Hayes modem on tty14; the 7300 uses the PC7300 internal modem. Both call the phone number 5551212 and use the datakit *Dialers* script to dial 9=5557777 using a Penril modem on the DATAKIT network. When *tarpon* answers the phone and gives a *login* prompt, the rest of the *Systems* script is used to login as *nuucp* with the *secret* password.

We hope that this section will help you setup UUCP, and that the example in Figure 21 may give you ideas on how you might customize your UUCP configuration to meet your users' needs. Just because some team members are not in the same building is no reason for them to be unable to communicate with any other project person via HoneyDanBer UUCP!

;login:

User-level Enhancements

User-level UUCP commands have not changed very much in the HoneyDanBer version. In this section we will show you a few of the user-level improvements we have found useful in our UNIX work.

We hope that you already appreciate the positive effects on end-user productivity provided by the administrator-level improvements described in previous sections. Our experience has shown that UNIX communities with good system administration are one to two orders of magnitude more productive than those with poor system support. Anything that makes UUCP reliability, availability, and serviceability easier for the system administrator is bound to have a high payoff for the entire user community.

One of the nicest "user-friendly" features of HoneyDanBer UUCP is its generalization of a consistent forwarding mechanism. Previously, most users assumed that the syntax used by **mail** was correct for **uucp** (see Figure 22).

For mail :	(correct)
mail	ihnp4!cbosgd!mrk
For uucp :	(not always correct)
uucp	file ihnp4!cbosgd!/usr/mrk/uucp/file

Figure 22: Forwarding Syntaxes in Old UUCP

Until System V, **uucp** could not handle this request.

In response to this syntactic dilemma, Mark Horton wrote the **uusend** command which was distributed with the Berkeley BSD releases. Like **mail**, a **uux** command was issued to execute **uusend** on each node in the path until the destination site was reached. However, if **uusend** was not one of the commands permitted to be executed by **uux**, the request failed midstream.

In the System V implementation, forwarding was a compile time option that allowed the **mail**-like syntax, but granted forwarding only to the public directory structure, */usr/spool/uucppublic*. However, this implementation required special hooks in the **uuxqt** command on the remote machine; hence, it failed if all of the nodes were not running the same release of UUCP.

HoneyDanBer **uucp** does not treat forwarding as a special case on the remote machine, rather it generates an appropriate **uux** command on the local machine. However, one restriction still remains: remote machines must allow **uucp** to be remotely executed.

Fortunately, this is an easy restriction to work around. Furthermore, since it is not always desirable to allow forwarding through your site (e.g., you might not want to forward other machine's files to sites in Australia), HoneyDanBer can restrict which machines can remotely execute the **uucp** command using the same mechanism it uses for any other command; no longer does the UUCP administrator need to set up special files to grant or deny forwarding permissions.

;login:

```
$ uuto binary.file opus!oliver!jrw

$ uustat
opusN752a 05/25-12:04 S opus wdr 1123 /u/wdr/hdb/binary.file
          05/25-12:04 S opus wdr uucp -C -njrw binary.file
                               oliver!~/receive/jrw/mtkam/
```

Figure 23: HoneyDanBer Remote **uuto**

Figure 23 shows how easy it is to keep in touch with project developers at remote sites. Using cross-compilers on a fast machine the object code can be transferred to the target machine for installation. Another nice "user-friendly" feature of HoneyDanBer UUCP is its intelligence to copy owner-access-only files into */usr/spool/uucppublic* for later transmission. The default is to transfer directly from the source file rather than make a duplicate copy in the spool area. However, **uucico** would not be able to open a protected file for reading, so **uucp** makes a *uucp*-owned copy when it is running with owner permissions.

Many of the internal functions of UUCP, such as lock file handling, modem dialing, and X.25 packet drivers, have been generalized for use in other applications. A side benefit of this is that **cu** can now search the *Systems* file to find the telephone number to dial. It also has a "-d" option so a user can watch the dialing progress to determine the cause of connection problems. Figure 13 showed an example of the "-d" option. Figure 24 shows how a user might capture the dialing progress for later analysis by the system administrator.

```
$ cu -d tarpon | tee tarpon.tee          < see Figure 13 >
(tarpon) login: wdr
Password:
LAST LOGIN: Tue Jun 10 08:22:09 1986
    Millions aren't working, but thank goodness they have jobs.
    ...happy grepping!

tarpon++ mail
mail: no space for temp file

tarpon++ df
/u10      (/dev/dsk13):    12140 blocks    461 i-nodes
/supt     (/dev/dsk03):    14396 blocks    901 i-nodes
/usr      (/dev/dsk11):    5640 blocks    247 i-nodes
/         (/dev/dsk00):      0 blocks     33 i-nodes

tarpon++ lf /tmp
Ex10352    dvia00783      poll.11111111      test
Rx00706    implog        ppsrvr10186/      test3.1520
acctoff    ipslog        sa.adrfl          tu

tarpon++ rm /tmp/*
/tmp/Ex10352: 600 mode? y
/tmp/Rx00706: 600 mode? y
/tmp/acctoff: 664 mode? y
    ...
```

;login:

```
tarpon++ mail
From uucp Fri Jun 13 20:10 EDT 1986
>From shaw Fri Jun 13 20:09 EDT 1986 remote from minnow
...
tarpon++ exit
```

results? check UNIX mail; /tmp probs – no space on root!

Figure 24: HoneyDanBer **cu** *system-name*

In this example /tmp ran out of space because an automagic backup utility initiated from **cron** was unable to **mount** a backup file system – so the root file system had one week's worth of file backups in one of its subdirectories! We quickly corrected the problem upon our return from the USENIX Conference.

A similar accident had happened before leaving for Atlanta. Somehow /usr/mail/wdr got **chowned** to **root** and we saw the error message shown at the top of Figure 25. It is surprising that the remote system got a better error message from HoneyDanBer's **uucp** command than **mail** gave to the local user!

On **mtkam**:

```
$ mail
mail: permission denied!          <--- WHAT'S THIS??
```

Later that day, on **tarpon**:

```
From uucp Mon Jun  9 14:40 EDT 1986
>From uucp Mon Jun  9 14:15 EDT 1986 remote from opus
>From uucp Mon Jun  9 14:20 EDT 1986 remote from mtkam
remote execution [uucp job mtkamN22dc (6/9-14:20:05)]
  rmail wdr
exited with status 1
```

```
=== stderr was ===
mail: cannot append to /usr/mail/wdr
```

Login to **mtkam** and fix it:

```
$ ls -l /usr/mail/wdr
-rw----- 1 root root 0 Jun  9 05:58 /usr/mail/wdr
```

<su to root and change /usr/mail/wdr>

```
$ ls -l /usr/mail/wdr
-rw-rw---- 1 wdr mail 0 Jun  9 05:58 /usr/mail/wdr
```

Figure 25: HoneyDanBer UUCP Error Messages Revisited

If you are sending lots of mail to project team members at remote sites, sometimes you might want to double-check that all the players were sent a letter. Since **mail** to a remote site actually invokes the **uux** command, you can check the /usr/spool/uucp/.Log/uux directory as shown in Figure 26 to make sure you didn't forget anyone.

;login:

```
$ pwd
/usr/spool/uucp/.Log/uux

$ ls -CF
MCNEILPH      opus      rruqq
oliver        pyuxc     tarpon

$ cat pyuxc
wdr pyuxc pyuxcN3ba5 (6/27-7:53:21,3930,0) QUEUED (rmail csaltz )
wdr pyuxc pyuxcN3ba6 (6/27-7:58:40,3935,0) QUEUED (rmail mdk )
```

Figure 26: Checking That Letters Were Mailed

Another useful facility is to be able to cancel letters before they are sent, in case you want to send a different version of an updated note. The `uustat -k` command allows you to kill a letter by job number, if it has not already been transmitted. Figure 27 shows how you can cancel many letters all at once.

```
# pwd
/usr/spool/uucp/tarpon

# ls -lt
total 50
-rw----- 1 uucp mail 2069 Jun 29 12:09 D.mtkam2a9add8
-rw-r----- 1 uucp mail 125 Jun 29 12:09 C.tarponN1d95
-rw----- 1 uucp mail 139 Jun 29 12:09 D.tarpo1d95de7
-rw----- 1 uucp mail 2069 Jun 29 12:09 D.mtkam2a99fad
...
-rw----- 1 uucp mail 140 Jun 29 11:44 D.tarpo1d940f0
-rw----- 1 uucp mail 2875 Jun 29 11:44 D.mtkam2a982b6

# >noon ! create a dummy file

# touch 06291200 noon ! give it the right time

# find . -newer noon -exec rm -f {} \; ! cancel afternoon mail

# ls -lt ! verify cancellation
total 8
-rw-r--r-- 1 root bin 0 Jun 29 12:00 noon
-rw----- 1 uucp mail 140 Jun 29 11:44 D.tarpo1d940f0
-rw----- 1 uucp mail 2875 Jun 29 11:44 D.mtkam2a982b6

# rm noon ! remove helper file
noon: 644 mode ? y
```

Figure 27: Canceling All Afternoon Mail

Notice that the example shown in Figure 27 is the first one that required us to use *super-user* privileges! That's because once your letters are in the **mail** system you no longer own them. "`uustat -k`", of course, would allow you to kill one of your own letters because it reads your user-id in the *C*. file. Again, the example shown in Figure 27 is a handy alternative to typing beaucoup many "`uustat -kjobid`" commands, if you have lots of letters to kill. However, this example was included for pedagogic reasons more than for its widespread practical use.

;login:

Whether or not you can make use of the examples in this paper depends on your particular needs and how stingy or generous your system administrator is with file permissions. We hope that our examples have helped you make better use of the UNIX System, particularly its UNIX-to-UNIX Communication Subsystem!

Part Two Summary

Figure 28 gives a brief description of several commands provided in the HoneyDanBer UUCP package. It was not our intent to regurgitate the manual pages for you, rather we wanted to show you some of the usual and unusual situations we have encountered in our UNIX systems work. We hope you have learned something useful from our paper and that it helps you in *your* work.

User Invoked Commands

uucp	file transfer command.
uuto	send files to a remote user.
uupick	retrieve files sent via uuto .
uux	remote command execution spooler.
uulog	displays UUCP logging information.
uustat	displays the status of previous UUCP commands, as well as allowing for the cancellation of UUCP jobs that have not yet been executed.

Administrative and Daemon Commands

uucico	UUCP networking daemon, performs the copying in and copying out of files between machines.
Uutry	starts a uucico between machines and places debugging information into a file and tails it.
uuclean	an <i>intelligent</i> cleanup command.
uucheck	a command that performs sanity checks on the local UUCP system.
uusched	deals with all the scheduling aspects of UUCP. It calls uucico with the name of the machine to call so that uucico does not need to search the UUCP directories for work.
uuxqt	the remote command execution server, executes commands spooled by uux .

Figure 28: HoneyDanBer UUCP Commands

Author Biographies[†]

Bill Rieken and Jim Webb are two professional triathletes* who enjoy UNIX as much as Johnny Carson enjoys the Tonite Show. They founded ".sh consulting" in 1984, specializing in UNIX system training and consulting for several clients worldwide, including AT&T Bell Laboratories. Both have considerable expertise in several UNIX systems running on a variety of computers. They are porting UNIX to run on a 32100-based fault-tolerant multi-computer system.

* triathlete (n.) – it normally requires three people to use UNIX: one to operate a terminal, another person to read the UNIX manual, and someone else who has done it before to show the other two how to do it. A UNIX triathlete can do all three.

Acknowledgements

The authors would like to thank Peter Honeyman, David Nowitz, and Brian Redman for taking time out of their busy schedules to review this paper. Although they were probably bored to tears reading about code they had written three years ago, their comments were very kind and helpful to us.

A special thanks is owed to Rich Mayer, who never read a UNIX manual in his life but types very fast, for helping us "idiot proof" some of our examples. His motto regarding UNIX manuals is: "Don't read the words. Just look at the pictures and examples. Get on a terminal and TRY IT!"

Others who helped are Edmond Gong, Wilton Chen, and Sam Torres, who nervously let us experiment with HoneyDanBer on their machines. Edmond provided the most help, although we won't tell him how. We'd also like to thank Dr. Rick Welsch, whose careful critique of the professional quality of our writing reminded us to forget about ever writing for UNIX/World.

Without the helpful cooperation of these people, the authors would probably still be sipping sugar magnolias at the Jersey Shore, while daydreaming about Big Sur and Laguna Beach, and pondering whether HoneyDanBer UUCP will *really* help UNIX fit on a Dick Tracy wristwatch someday.

References

1. *AT&T 3B2 Computer UNIX System V Release 2.0 Basic Networking Utilities Guide*, (Select Code 305-432) Issue 1, October 1984.
2. *UNIX System V Administrator Guide – DEC Processors*, (Select Code 307-101) Issue 1, December 1983.
3. Version 2 and HoneyDanBer UUCP source code: */usr/src/cmd/uucp*, */usr/src/cmd/cu.c*, */usr/src/cmd/ct.c*.
4. "An Interview with Peter Honeyman," *UNIX Review*, January, 1986.
5. "A Dial-Up Network of UNIX Systems," D. A. Nowitz and M. E. Lesk, "UUCP Implementation Description," D. A. Nowitz, *UNIX Programmer's Manual, Volume 2* Holt, Rinehart and Winston.
6. "Sendmail Revisited," Eric Allman and Miriam Amos, *Summer 1985 USENIX Conference Proceedings*.
7. "Loving UUCP, or How I Spent My Summer Vacation," "Experimental Implementation of UUCP – Security Aspects," Honeyman, Nowitz, and Redman, "Mapping the UUCP Network," Rob Kolstad, Karen Summers-Horton, *1984 UniForum Conference Proceedings*.
8. "UUCP and Netmail," Karen Summers-Horton, Mark Horton, Peter Honeyman, Pat Parseghian, Michael O'Brian, and Raymond Essick, *Winter 1985 USENIX Conference Proceedings*.
9. "UUCP Mapping Project," *Summer 1984 USENIX Conference Proceedings*.
10. "A Perspective on the USENET," Eric Fair, *USENIX ;login:*, January/February, 1986.
11. "PATHALIAS or The Care and Feeding of Relative Addresses," Peter Honeyman and Steven Bellovin, "Mail Routing using Domain Names," Craig Partridge, "AT&T Mail," Dale DeJager, "A Mail File System for Eighth Edition UNIX," David Hitz and Peter Honeyman, *Summer 1986 USENIX Conference Proceedings*.

[†] The following is how the authors describe themselves. -Ed.

;login:

Netnews Under VM/CMS

Irwin Tillman†
6040133@pucc.bitnet

Peter Honeyman
princeton!honey

Department of Computer Science
Princeton University
Princeton NJ 08544

ABSTRACT

We describe an implementation of *netnews* under IBM VM/CMS. Differences between CMS and UNIX® lead to changes in the organization of the data and access procedures. Written with command language and editor scripts, our implementation includes database maintenance and a screen-oriented browser. Public distribution is planned.

Goals

Netnews is a popular bulletin board running on several thousand UNIX-based machines. With over 250 newsgroups, each devoted to a topic of discussion, *netnews* caters to technical interests, and to a variety of recreational topics as well. The software was written in the late 1970's by a group of graduate students at Duke and UNC and has been modified by others; it is in the public domain, and may be used by any UNIX machine with links to a neighbor running *netnews*. Each computer forwards news to neighboring machines, avoiding loops by using path information in the individual articles. News flows slowly through the network, each article radiating out of its originating machine. This loose network is called USENET.

Our goals were to implement *netnews* under VM/CMS and to provide the end user with a facility similar to *vnews*, a CRT-oriented news browser. The documentation supplied with the 2.10.2 distribution of UNIX *netnews* software served as our guide.¹ CMS is IBM's most popular timesharing operating system, providing a conversational environment for application software development and use.

Implementation

Netnews software can be divided into two parts. The first is a set of programs that maintains articles on a local machine, sends articles to a neighbor, and installs new articles from remote machines and local users; in short, it performs database functions. The second group of programs provides the user interface to the database and is largely independent of the database software; in fact, several user interfaces can generally be found on UNIX machines. For our initial implementation, we designed a visual interface similar to *vnews*, which uses single-key functions to examine the database.

The Visual Interface

We selected XEDIT to provide a framework for our visual interface, which we call *readnews*. XEDIT, IBM's CMS editor, is fast and allows a great deal of customization of its appearance; it also hides the details of screen management from *readnews*. XEDIT can be made to appear to be a newsreading program with the use of a REXX macro to intercept the user's input. What would be written in shell scripts under UNIX is written in REXX in CMS.

REXX is not a command level interpreter like the UNIX shell; you cannot type REXX statements to CMS. It is an interpreted language similar to Pascal, used to build new commands out of sequences of existing ones. The language is designed to provide a uniform interface to a number of environments, including XEDIT.²

† This work was supported by the Princeton University Computing Center and the Computer Science Department.

¹ See in particular Mark Horton's *Standards for Interchange of USENET Messages*.

² For an introduction to REXX, see M.F. Cowlshaw's *The REXX Programming Language: A Practical Approach to Programming*, Prentice Hall, 1985.

;login:

The Database

We mentioned that the software maintaining the local news is in essence a database manager, thus we considered using a general DBMS to handle the articles,³ but discarded this approach for several reasons. First, the organization of the data is simple, so that the overhead involved could not be recouped in faster retrieval times. Second, it made our implementation depend on a specific DBMS, not generally available at most CMS sites. Third, by adding a distinct and complex subsystem with which few CMS programmers are familiar, it made the software much more difficult to maintain and modify.

Instead, we use REXX for the database functions as well. This has resulted in a coherent package that does not require a database specialist to maintain and (we hope) is easily modified to suit local needs.

One drawback of this choice is in the performance of the database software. REXX is (currently) an interpreted language, therefore the maintenance programs do not run as fast as compiled software. Nonetheless, the REXX interpreter is fast and uses specially designed CMS routines to access files.

CMS File Management

One immediate difference between CMS and UNIX is in file management. CMS does not provide a hierarchical file system, so useful in UNIX. Instead, each of a user's "minidisks" is independent, with no partitions into smaller regions. Files are identified not by a pathname, but by a triple (*user*, *disk*, *fileid*). Consequently, the user organizes a disk by selecting appropriate *fileids* composed of a *filename* and a *filetype*. (Each is a maximum of eight characters.) Limited pattern matching is available to process groups of files with similar fileids.

This style of organization is not conducive to the UNIX implementation of *netnews*, where article 2847 in newsgroup *net.games.chess* might be contained in */usr/spool/netnews/net/games/chess/2847*. The problem is easily remedied, but with a considerable loss of clarity. Each newsgroup is encoded as a four-digit number, which is used as the article's filename. The article's local sequence number within the newsgroup is used as the filetype. Thus, if the group number of *net.games.chess* is 0213, the same article is contained in *0213 00002846*; to locate all the articles in the newsgroup we use *LISTFILE 0213 **. This implementation permits up to 10,000 newsgroups and 100,000,000 articles within each newsgroup.

Allowing users to share access to constantly changing files is more difficult in CMS than in UNIX. A user can define a number of minidisks containing files; each minidisk may be made available to other users on a read or read/write basis. Allowing more than one user simultaneous write access to the same minidisk virtually assures a loss of data. We therefore establish one user with read/write access to a shared disk and require updates by other users to be performed by this "server account." All the articles and associated tables are located on a publicly readable disk, which is writable only by the server.

A related problem arises from the CMS view of a read-only disk. When a user requests read-only access to a disk, a private *user file directory* of the disk is created for him. If the server subsequently modifies the disk, the user's file directory does not reflect the true state of the disk, which may result in read errors. We avoid this problem by defining two transaction types for manipulation of the database by the server:

- The server may create new files. A user does not see files created since she last accessed the disk.
- The server may append or change data within *update-in-place* files. (Changes in these special files are written back to the disk in the same location from which they came.) Again, a user does not see any changes made since her last access.

Restricting ourselves to these two actions leads to the following guidelines:

- We may add articles at any time.
- Our *expiration* and *contents* files must be update-in-place files, and may be updated at any time, provided that we do not shorten these files.
- We hold control messages that might remove articles from the disk or text from the *expiration* or *contents* files until we disallow user access to the database.

³ The DBMS we considered is SPIRES, an extensive package developed at Stanford.

;login:

These guidelines summarize the operation of the server account. The server may add articles at any time, updating in place the *contents* file, reflecting the articles available in each newsgroup, and the *expiration* file. Users see news current through the last time they accessed the disk, and all top-level user-callable routines begin by reaccessing the news disk, which builds a new user file directory. Destructive operations are done overnight. We lock the disk, process control messages that accumulated during the day, and expire old articles. While the disk is locked, user routines refuse to run. Users can easily bypass these programs and access the news disk by hand, but they do so at their own risk and pose no danger to the database.

The design of the server is straightforward. The server is an account that is always running with write access to the database disk.⁴ Communication with other users is handled through the server's "virtual card reader." The reader is a private location belonging to the server to which any user can send files; no other users have access to these files, and each file is stamped with the sender's name. Using the reader mechanism supplied by the system hides the locking issues and guarantees the validity of user identification. The posting software sends articles to the server's reader, which the server retrieves and posts to the database.

The news administrator can halt the server gracefully by sending a special file to its reader; this allows the administrator to log on to the server account when necessary. The administrator's account may also send the standard control messages this way (add or remove a group, check the groups, etc.) and also request a copy of the server's console recording. Requests to cancel articles are also handled by sending a special file to the server, which authenticates them and processes them in the wee hours. Because the format of these files is extremely simple, users can write their own *postnews* program if they don't like ours.

Communication with Other Computers

No mention has been made of how we talk to the outside world. This is because there is no single answer; we anticipate that communication methods vary greatly from site to site. Instead, hooks in the server account facilitate local customization. Outgoing articles are batched periodically by the server and passed to a program specified locally. Ideally, the computers with which news is exchanged are directly connected to the local machine, so that articles can be mailed to the appropriate news-reading programs on neighboring machines.

For example, our IBM 3081, called *pucc*, and the VAX that feeds us, *princeton*, do not directly talk to each other; both are connected to *puvax2*, but *pucc* has no mail connection. The 3081 uses *fip* to transfer a batched news file to *puvax2*, which *uux*'s the file to the appropriate program on *princeton*.

Similarly, incoming articles are easily handled if the computers are mail-connected. If a computer can uniquely identify mail sent from the news account on a feed machine, the feed can mail articles to a local server, which retrieves them from its virtual card reader and processes them the way it does local submissions. Again, we have no mail connection, so *princeton* *uux*'s the files to *puvax2*, which *fip*'s them to a disk on *pucc*. The server periodically checks this disk for incoming news.

Future Work

In addition to *readnews*, the user may run *postnews*, *checknews*, *cancel* (delete an article), and *tossnews* (discard all current news as read). We are also preparing an archiving mechanism that saves articles from specified newsgroups permanently. Additionally, the key subroutines used by these programs are provided in stand-alone form to allow users to write their own interfaces to the news database if they desire.

We have designated this implementation *version P 1.0*. Unquestionably there is much room for improvement; our primary goal has been to get *netnews* running at all, leaving sophisticated features for later development. Our first such concern is to establish mail links with our news feed, to make the communication portion of the server software mesh better with the remainder of the package. Of course, we have a wish list of features for *readnews* as well.

Version P 1.1 should become available for testing at other sites during the Fall of 1986; interested parties should contact usenet@pucc.bitnet for more information.

⁴ At boot (or IPL) time, it logs on to this account, initiates an endless loop, then disconnects the account, allowing it to run with no associated terminal. This is analogous to a daemon process in UNIX.

Reflections On A UNIX® Scheduler

Thomas A. Bohannon

Harris Corporation
Business Information Systems Operations
Systems Development
1704 Chantilly Dr. NE, Suite C
Atlanta, Georgia 30324
{gatech,akgua}!galbp!tab

ABSTRACT

Modifications made to an existing UNIX scheduler have resulted in a simple, straightforward scheduler that gives significant performance gains with a minimal amount of kernel changes. As currently implemented on UNIX Version 7 and UNIX System III ports, the scheduler has increased interactive response from 31%-87% under varied load conditions. Far from perfect, the changes appear to be a step in the right direction. Every attempt was made to enhance the simplicity and elegance familiar in the UNIX programming environment. The three main areas of discussion include the traditional UNIX scheduler, the changes made, and the results of the changes. The traditional scheduler is discussed briefly along with the change in the nature of most applications that necessitated the scheduler change. Second, the concepts behind the new scheduler along with the actual kernel changes are discussed. Thirdly, the results of the changes as they affect particular processes as well as the overall system performance are discussed.

1. Primary Goals

Reworking the existing scheduler had several primary goals. The system should provide good interactive response and a way to keep any one process from monopolizing the system resources. Any given process should not be able to consciously alter the scheduling mechanism. Perhaps most importantly, however, any changes should be kept to a bare minimum and as simple and straightforward as possible to remain well within the realm of the spirit of the UNIX environment.

The system should give good interactive response, which generally means that a user should not be able to outpace the character echo of a process functioning in either raw or cooked mode. Before full screen editors such as *vi*, nearly all utilities executed in cooked mode relying on the terminal drivers to echo typed characters. A process awaiting terminal input would block itself until a full line of input was typed. Only then would a task switch occur giving the process an opportunity to receive the incoming data. However, in a process executing in raw mode a task switch must occur for every incoming character allowing the process to receive and echo the character if needed. Generally this means that interactive processes need only a small amount of CPU time at any instant, but they need it as soon as possible.

A process should not be able to monopolize the system resources. One way this condition can occur in the standard scheduler is in a process that has a high ratio of system time to user time. The longer a process is executing kernel code, the greater the chances that it encounters a critical region and temporarily disables interrupts. More importantly, as long as a process is executing in kernel code and not blocked on a resource, a task switch can not occur to allow a preempted process to continue executing until it exits system mode.

A scheduler that allows a user process to alter its own behavior within the system immediately hinders program portability to other versions of UNIX. Resource allocation should be controlled solely by the operating system. To maintain transparency to user processes, no system or function calls should be added. A process should not have any means to alter the scheduling mechanism except the *nice(2)* system call that is found on all UNIX systems.

Any changes to the kernel should be minimal and kept as simple as possible. The UNIX scheduler is generally known to be a target of kernel hackers and tends to be the testing field of various new academically inspired algorithms. Scheduler changes can quickly become unbelievably complex. When this condition occurs

;login:

the simplicity and elegance of UNIX quickly breaks down. The Berkeley 4.[23] scheduler is a good example of an idea gone awry.[1]

2. Traditional Scheduler

The traditional UNIX scheduler gives every process a fixed quantum of one second and uses a basic round-robin scheduling mechanism. A compute to real-time ratio for each process is recalculated every second. This ratio is the main value used in calculating a process's priority. The longer a process runs the larger its ratio becomes causing it to have a lower priority. On the other hand, the longer a process waits for the opportunity to run, the smaller the ratio becomes causing it to have a more favorable priority. When a process awakens after waiting for some system event to occur it is assigned a good initial priority that usually causes it to preempt the current running process. [2]

As the nature of the processes that run in the UNIX environment changes, the requirements of the scheduler change. Originally, most interactive processes executed in "cooked" mode meaning that the serial device drivers were responsible for all canonical processing. Incoming characters were placed on a queue and were not given to the process until a newline was received. As more and more interactive applications are written for the UNIX environment the system resource demands for "raw" mode processing increases. Raw mode simply means that as each character is received it is given to the process to perform the canonical processing. A good example of cooked and raw processes are the UNIX editors **ed** and **vi**. **ed** is a line editor which executes in cooked mode whereas **vi** is a full screen editor which executes in raw mode.

3. New Scheduler Implementation

3.1. Data Structure Changes

Very few changes were made to the existing structures within the kernel and only one new structure was added. The only existing structure modified was the process table. Three additional members were added to the process structure:

```
short p_quantum; /* cumulative quantum value */
char p_person; /* personality type */
char p_ratio; /* ratio of stime to utime */
```

The *p_quantum* member contains the current scheduling quantum of the process measured in clock ticks. The *p_person* member contains a value associated with a particular personality. This value is really a subscript into another table that will be discussed later. The *p_ratio* member contains the current value of the process's total system time divided by total user time, measured in clock ticks.

A new structure was added that defines various bounds according to the personality of a process. Currently there are four possible personalities (cpu, disk, tty, pipe) each with its own characteristic scheduling behavior.

```
struct vsched {
    short m_cpu; /* maximum cpu clicks allowed */
    short m_cpuinc; /* p_cpu increment */
    short m_quantum; /* p_quantum increment value */
    short m_divisor; /* divisor for p_cpu decay */
    short m_cum_quan; /* max. cumulative quantum size */
} vsched[4];
```

The member *m_cpu* gives the maximum value that the *p_cpu* value can accumulate. *m_cpuinc* is the amount by which *p_cpu* is incremented every clock tick that the process is caught with control of the CPU. *m_quantum* is the quantum size added to *p_quantum* each time a process is given control of the CPU. The member *m_divisor* is used in the decay of the *p_cpu* value while a process is not in control of the CPU. The member *m_cum_quan* is the highest value that a process's quantum is allowed to attain.

;login:

3.2. Process Personality Types

Processes are assigned one of four personalities (*cpu*, *tty*, *disk*, or *pipe*). All processes are initially classified as CPU intensive until they prove themselves otherwise. As a process's behavior changes, its personality value will vary. The purpose is to allow the scheduler to favor one type of process over another type. Flexibility is given to the system administrator to categorize the four possible types of processes from most favorable to least favorable.

A hook was placed in *rdwr()* to set the personality byte to *tty* if the process is accessing a character device, to *disk* if the process is accessing a block device, and to *pipe* if the process is accessing a pipe or fifo. The *rdwr()* routine was chosen as the site for the hook placement for simplicity: the desired effect could be achieved by changing only one routine.

At the end of a process's quantum the high bit of the personality byte is examined. If set then the personality has been reconfirmed during the quantum just completed. As a result, the process is allowed to keep its current personality. If on the other hand the high bit is not set, then its current personality has not been reconfirmed during the last quantum and is therefore reassigned to *cpu*. This prevents, for example, a CPU intensive process from initially performing a *printf()* to gain a *tty* type personality. As long as the process is performing terminal output it will have a *tty* personality, but as soon as it becomes *cpu* intensive its personality type will change accordingly. The resources that a process receives are based on the personality type.

The CPU usage variable for each process is not allowed to grow higher than the allowable limit for its personality. Each time the process's CPU usage variable is incremented it increases by the amount found in the increment value of the given personality. The quantum size is added to the process's quantum (not to exceed the maximum cumulative limit) each time the process gains control of the CPU. This allows a process to save the remaining portion of its current quantum for future use if it blocks itself or is preempted by another process. The cumulative limit is set to prevent a process from temporarily hogging the system resources with an enormous quantum. The CPU usage variable decays while a process is not running so its priority will gradually become better. The rate at which the CPU usage decays is a function of the divisor variable which can differ according to the personality, thus allowing varying decay rates for each personality type. If one wishes to penalize a pipe intensive process more than a CPU intensive process he might assign a larger CPU usage divisor to the pipe type that will cause the CPU usage value to decay at a much slower rate than the CPU intensive process. The net result is that the priority of the pipe intensive process remains low (higher value) for a longer period of time.

3.3. Quantum Management

A quantum is a slice of processor time given to a process during which it executes. Since the UNIX scheduler is basically a round-robin algorithm, large quanta tend to make it act as a first come first serve (FCFS) scheduler. A process will execute until it completes or blocks itself. When small quanta are chosen the system resources are more evenly distributed among the runnable processes. Small quanta are favorable to interactive environments allowing quicker character by character processing for programs executing in raw mode giving users better interactive response. If quanta too small are chosen, then more time could be spent in system mode performing task switches than actually running user processes.[3] The bulk of the quantum management takes place within the clock interrupt handler. The only other routine dealing with quantum management is *swtch()*. Just before a process is set to run, its quantum size is increased according to its personality (not to exceed the given limit).

The following code was added to the clock interrupt handler to be executed at every clock tick.

;login:

```
if ( pp != &proc[0] ) {
    struct    proc *ap;

    if ( pp->p_cpu < vsched[pp->p_person&VS].m_cpu )
        pp->p_cpu += vsched[pp->p_person&VS].m_cpuinc;

    pp->p_ratio = (u.u_stime / ( (u.u_utime)?u.u_utime:1 ))&0x7f;

    if ( --pp->p_quantum <= 0 ) {
        if ( pp->p_quantum < 0 )
            pp->p_quantum++;
        pp->p_person = (pp->p_person&VSCHG) ?
            pp->p_person&VS : VSCPU;
        runrun++;
        for ( ap = &proc[1]; ap <= &proc[Nproc]; ap++ )
            if ( ap->p_stat )
                setpri(ap);
        curpri = pp->p_pri;
    }
}
```

At each clock tick, if the current running process is not the swapper then perform quantum management tasks. If a process has not exceeded the maximum amount of CPU usage allowable, then increment its usage value by the increment variable for the process's personality. Priority calculations are based on the CPU usage value. However, there is a point where the scheduler no longer cares the exact amount of CPU usage but only that the process is using a lot. Priority calculations are also based on the ratio of system time to user time and is calculated every clock tick for the current running process.

Once a process's quantum expires, its personality is verified, the priority of all processes is recalculated, and a task switch happens. During each quantum a process must reassert its personality or it will default back to CPU.

Code still remains to calculate priorities every second for each process. Generally all lightning bolt processing is left undisturbed.

3.4. Setting Processes' Priority

It seems that Version 7 UNIX contains a function called *setpri()* that calculates the priority of a given process whereas UNIX System III does not. The basic idea of the *setpri()* function remains unchanged with the exception that the CPU usage value is decayed **after** it is used in the priority calculation.

```
setpri(pp)
struct    proc *pp;
{
    pp->p_pri = pp->p_cpu + PUSER + pp->p_nice - NZERO + pp->p_ratio;
    pp->p_pri &= 0x7f;

    if ( pp->p_cpu && pp != &u.u_procp ) {
        pp->p_cpu -= (pp->p_cpu/vsched[pp->p_person&VS].m_divisor+1);
        pp->p_cpu = (pp->p_cpu < 0)? 0 : pp->p_cpu;
    }

    if ( pp->p_pri < curpri )
        runrun++;

    return ( pp->p_pri );
}
```

3.4.1. Priority Calculation

The priority of a process is the sum of its total CPU usage, a constant (PUSER 50), the nice value, and the ratio of system to user time.

The purpose of NZERO, which is 20, is to serve as the baseline value for nice. A normal process has a nice value of 20 with a net result in the priority calculations of 0. A super-user process may nice itself down by

;login:

setting nice to zero which decreases the priority value by 20 resulting in a more favorable priority. On the other hand, a considerate process may nice itself by setting nice to 40 increasing the priority value by 20 resulting in a less favorable priority.

The CPU usage value is a multiple of how many times the clock interrupt occurs and catches the process with control of the CPU. The PUSER constant is a baseline value. The best priority that a process can ever hope to have as a result of calling *setpri()* is PUSER, unless, of course, the process has been niced in which case the priority could be PUSER-NZERO (30). Note that a process can have a priority lower than PUSER or PUSER-NZERO only as a result of going to sleep waiting on an event such as disk I/O.

The idea behind the ratio of system to user time is to punish a process as it spends more time in system mode than in user mode. While a process is executing in system mode, no task switch can occur unless the process blocks itself or returns to user mode. A process awakening from a blocked state will usually have a better priority than the current running process. For example, if an interrupt occurs signaling some I/O complete, enabling a process with a priority less than the current running process, a task switch cannot occur until the current process exits from system mode.

3.4.2. CPU Usage Decay

As long as a process is executing, its CPU usage field grows thus causing it to have a progressively worse priority. On the other hand, when a process is not executing its CPU usage field decays causing its priority to gradually become better. The rate at which the value decays is a key element in calculating the process's priority. Each personality has associated with it a divisor value by which the CPU usage value is divided. The result is used to decay the process's CPU usage value. The general effect is that the CPU usage decays rapidly at first and eventually levels off to become zero.

The idea is that as a process executes, its priority becomes worse until its quantum expires and a task switch occurs. As the process awaits its next quantum, its priority becomes more desirable. When the next task switch occurs, the process with the best priority of all runnable processes is chosen.

3.4.3. Preempting Process

Whenever a process's priority is calculated within *setpri()* it is checked against the priority of the current running process. If the priority of the current running process is found to be worse than that of the process just calculated, the task switch flag *runrun* is set causing *switch()* to be called at the first available opportunity.

3.5. Scheduler Tailoring

Obviously the values of the vsched structures play a crucial role in the behavior of the scheduler. Experimentation was done by changing the initialization values of the structures and relinking the kernel. This quickly became a slow and monotonous process. Therefore a pseudo-device was created to allow on-the-fly change of the vsched structure.

The scheduler device (*/dev/sched*) is a pseudo-device that allows behavior change of the scheduler without relinking and rebooting the kernel. A *read(2)* of the device returns the contents of the vsched array of structures and a *write(2)* to the device replaces the contents of the structures, thus allowing instant change of the scheduler. The device was initially created only as a means for testing and tuning of the scheduler. However, the scheduler pseudo-device appears to have several positive side effects that justify its remaining an integral part of the new scheduler. It might be very beneficial, for example, to have a daemon modify the scheduling mechanism in the evening hours to favor pipe or compute intensive processes and to discourage interactive processes. On a broader scope, the ease of varying the scheduler's behavior allows individual installations to fine tune the kernel for a particular environment.

One must realize that a lack of discretion by the system administrator concerning the pseudo-device can have negative consequences. However, the same holds true for other devices and particularly for the root filesystem device! With a little common sense, */dev/sched* can be a great asset in obtaining the maximum horse-power available from the system.

;login:

4. Actual Experiences

The effects of the new scheduler have been far greater than expected. In attempting to increase interactive response under loaded conditions, some degradation of CPU intensive processes was expected. However, they improved by 7-8% and interactive processes improved from 31% under light load conditions to 87% under heavier load conditions! Before the scheduler modifications, interactive response degraded rapidly as the machine load increased. With the new scheduler, interactive processes show virtually no degradation as machine load increases.

4.1. Benchmarks

The benchmark consisted of a shell script that initiated dictionary **sort**'s (25,000 words) background and performed a **cat**(1) of the *termcap* file foreground. The **sort**'s and the **cat** both have a moderate amount of disk activity with the **sort**'s tending to be CPU intensive and the **cat**'s tending to be tty intensive. In lieu of **cat**'ing the *termcap* file, further tests were performed using a process with repetitive writes to the terminal and no disk access. It, too, showed performance gains in line with the previous tests.

4.2. Everyday Life

All too often, benchmarks are developed to prove some point and rarely measure general performance. With the new scheduler, the benchmarks discussed are not the only tests that show improvement. Normally, on our processor response in **vi**(1) is intolerable with just one or two **make**(1) commands running background. With the scheduler modifications, **vi** responds extremely well with four and five **make** commands running background and shows virtually no degradation!

A real example is a terminal emulation program that continuously polls both a serial port and the keyboard for incoming characters. The select type function spends a fair amount of time in system mode and unlike other I/O calls can never block itself. When this process was running, a login sequence from another port that would normally take 12-14 seconds took over 2 minutes! With the new scheduler the login sequence takes 15 seconds with one (and even more) of the emulation processes running. Furthermore, the data transfer effectiveness of the emulation processes remains high. Although the mentioned program is poorly designed, no one process should be able to monopolize the system resources.

4.3. A Comfortable Configuration

The following table illustrates the configuration of the system on which the mentioned experiences and tests were performed. A clock interrupt occurs every 25 milliseconds (40Hz).

CPU: Intel 80186 10MHz Memory: 1.5MB				
Variable	Personality Type			
	CPU	TTY	DISK	PIPE
m_cpu	70	70	70	70
m_cpuinc	5	5	5	5
m_quantum	4	2	3	5
m_divisor	4	2	3	4
m_cum_quan	4	4	6	15

4.4. UNIX Versions & Hardware

The scheduler modifications were initially made to a port of the UNIX System III kernel to the Intel 80186 microprocessor. Later, the same scheduler was applied to a port of the UNIX Version 7 kernel to the same processor and showed performance gains equivalent to those found with System III.

Although the modified kernel has only been tested on microcomputers, there appears to be no reason that similar performance gains could not be achieved on hardware of considerable more horsepower. Every attempt

;login:

was made to ensure that the scheduler not be overly complex or specific to any one type of hardware architecture.

5. Further Work

The hook placed in *rdwr()* to set the personality byte has several shortcomings. At this level within the I/O request many assumptions are being made that can conceivably be incorrect. For instance, just because a process is accessing a character device does not necessarily mean that the device is a terminal device. A process could be accessing a magnetic tape device via the raw device driver. Also at this level of the I/O request it is not known when accessing a block device whether the requested block is currently in the buffer cache. Therefore the process may or may not block itself awaiting completion of the I/O request. Perhaps a more exact distinction could be made between processes really performing terminal I/O as opposed to accessing a raw device and between I/O requests that really cause disk activity to occur as opposed to requests that are fulfilled directly from the buffer cache.

Modifications were made to the *ps(1)* command to display the three additional fields within the process table (*p_quantum*, *p_person*, *p_ratio*). This is somewhat helpful in observing process behavior. Because of the amount of processing time *ps* needs in order to locate and display the process table information, it is not a good tool for monitoring the effects of various *vsched* variables on process behavior. Perhaps a more useful observation tool would be another pseudo-device which, when read, would return proc table information for each process for a given number of consecutive clock ticks after the read request is initiated.

All of the scheduler modifications only address the problem of processor utilization and do not address the issue of process swapping. More investigation as to the impact of the modifications on the swapping procedures needs to take place.

6. Conclusions

All of the primary goals were achieved with the performance improvements being far greater than expected. The ability to change the scheduler's behavior via a pseudo-device was originally intended for initial tuning purposes only. However, after implementation, benefits other than those originally intended were seen, and as a result the drivers remain.

Attempting to solve the multitude of problems associated with scheduling is an overwhelming task that cannot be taken lightly. It is so easy to make changes that at first seem logical, but turn out to have unforeseen negative effects. Throughout the entire effort, every attempt was made to keep kernel changes to a minimum with the maximum amount of positive results. More importantly the author hopes not to have contributed to the corruption of simple elegant UNIX.

References

- [1] J. H. Straathof, A. K. Thareja, A. K. Agrawala, "UNIX Scheduling for Large Systems," 1986 Winter USENIX Technical Conference Proceedings.
- [2] K. Thompson, "UNIX Implementation," The Bell System Technical Journal, July-August 1978, Vol. 57, No. 6, Part 2.
- [3] D. C. Tsichritzis and P. A. Bernstein, *Operating Systems, Computer Science and Applied Mathematics*, Academic Press, 1974

Local User Groups

The USENIX Association will support local user groups in the following ways:

- Assisting the formation of a local user group by doing an initial mailing for the group. This mailing may consist of a list supplied by the group, or may be derived from the USENIX membership list for the geographical area involved. At least one member of the organizing group must be a current member of the USENIX Association. Membership in the group must be open to the public.
- Publishing information on local user groups in ;login: giving the name, address, phone number, net address, time and location of meetings, etc. Announcements of special events are welcome; send them to the editor at the USENIX office.

Please contact the USENIX office if you need assistance in either of the above matters. Our current list of local groups follows.

In the **Boulder**, Colorado area a group meets about every two months at different sites for informal discussions.

Front Range Users Group
N.B.I., Inc.
P.O. Box 9001
Boulder, CO 80301

Steve Gaede (303) 444-5710
hao!nbires!gaede

Dallas/Fort Worth UNIX User's Group
Seny Systems, Inc.
5327 N. Central, #320
Dallas, TX 75205

Jim Hummel (214) 522-2324

In the **Washington, D.C.**, area there is an umbrella organization called Capitol Shell. It consists of commercial, government, educational, and individual UNIX enthusiasts. For information and a newsletter write:

Capitol Shell
8375 Leesburg Pike, #277
Vienna, VA 22180

seismo!cal-unix!capish

In the **New York City** area there is a non-profit organization for users and vendors of products and services for UNIX systems.

Unigroup of New York
G.P.O. Box 1931
New York, NY 10116

In **Minnesota** a group meets on the first Wednesday of each month. For information contact:

UNIX Users of Minnesota
Carolyn Downey (612) 934-1199

In the **Atlanta** area there is a group for people with interest in UNIX or UNIX-like systems, which meets on the first Monday of each month in White Hall, Emory University.

Atlanta UNIX Users Group
P.O. Box 12241
Atlanta, GA 30355-2241

Marc Merlin (404) 442-4772
Mark Landry (404) 365-8108

In the **Seattle** area there is a group with over 150 members, a monthly newsletter, and a software exchange system. Meetings are held monthly.

Bill Campbell (206) 232-4164
Seattle UNIX Group Membership Information
6641 East Mercer Way
Mercer Island, WA 98040
uw-beaver!tikal!camco!bill

An informal group is starting in the **St. Louis** area:

St. Louis UNIX Users Group
Plus Five Computer Services
765 Westwood, 10A
Clayton, MO 63105

Eric Kiebler (314) 725-9492
ihnp4!plus5!sluug

;login:

In the **northern New England** area is a group that meets monthly at different sites. Contact one of the following for information:

Emily Bryant (603) 646-2999
Kiewit Computation Center
Dartmouth College
Hanover, NH 03755
decvax!dartvax!emilyb

David Marston (603) 883-3556
Daniel Webster College
University Drive
Nashua, NH 03063

A UNIX/C language users group has been formed in **Tulsa**. For current information on meetings, etc. contact:

Pete Rourke
\$USR
7340 East 25th Place
Tulsa, OK 74129

The **New Zealand** group provides an annual Workshop and Exhibition and a regular newsletter to its members.

New Zealand UNIX Systems User Group
P.O. Box 13056
University of Waikato
Hamilton, New Zealand

In the **San Antonio** area the San Antonio UNIX Users (SATUU) meet twice each month with the second Wednesday being a dinner meeting and the third Wednesday being a "roving" meeting at a user site.

San Antonio UNIX Users
7950 Floyd Curl Dr. #102
San Antonio, TX 78229-3955

William T. Blessum, M.D. (512) 692-0977
ihnp4!petrol!bles!wtb

A new UNIX users group is starting in the **Coral Springs**, Florida, area. For information, contact:

S. Shaw McQuinn (305) 344-8686
8557 W. Sample Road
Coral Springs, FL 33065

USENIX Association
P.O. Box 7
El Cerrito, CA 94530

First Class Mail

FIRST CLASS MAIL
U.S. POSTAGE PAID
El Cerrito, CA 94530
Permit No. 87

HoneyDanBer UUCP, Part 2

Netnews Under VM/CMS

Reflections on a UNIX Scheduler

Graphics Workshop Proceedings Available

Change of Address Form

Please fill out and send the following form through the U.S. mail to the Association Office at the address above.

Name: _____ Member #: _____

OLD: _____ NEW: _____

Phone: _____

uucp: {decvax,ucbvax}! _____